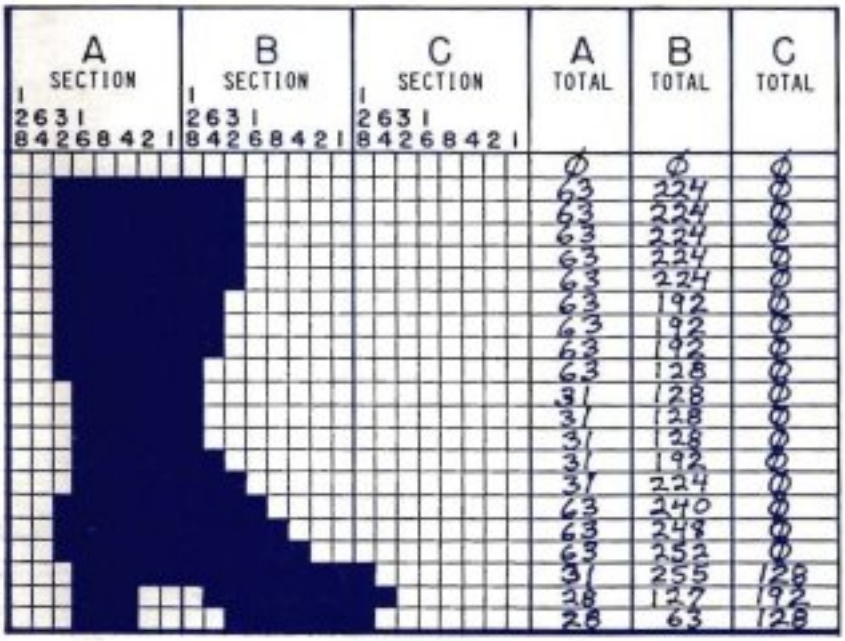
COMAL TODAY 22

Figure 3 - see page 11



Walking Sprites - page 11

COMAL Today
5501 Groveland Ter
Madison, WI 53716

If the label says
Last Issue: 22
You must renew now.
Use order form inside.

Bulk Rate
U.S.Postage
Paid
Madison, WI
Permit 2981

Figure 2 - see page 11

Best Books

Submit Articles and Programs

Send any COMAL information, programs, or articles that you would like to share to:

COMAL Users Group, U.S.A., Limited
5501 Groveland Terrace
Madison, WI 53716

If you submit a program, please send it on disk. A printed listing of the program is not necessary. If possible, also include a text file explaining the program. **Put your name as a remark at the beginning of your programs.** This helps us give proper credits if they are used. Most important: label the disk with your name, address and date. Also include disk format: C64, IBM, CP/M, Apple, etc.

Articles should be submitted as standard text files on disk. If possible, also include a printout of each text file on the disk. Don't include any special formatting commands in your files (we have to delete them). We use special formatting for our LaserJet printer. (We use Word Perfect 4.2 on our Zenith IBM compatible).

Don't worry if you aren't a professional writer. We try to keep this newsletter informal. The information and programs are more important than perfect grammar. Articles sent to us go through extensive editing. We actually go through over 4,000 sheets of paper while preparing one newsletter! You don't have to follow a bunch of rules, either. We rework your submissions to fit our newsletter format.

Material submitted is not returned. However, if we use your material, we will send you a copy of the newsletter/disk it is in or extend your subscription. ■

The popular reference book *COMAL Handbook* is now out of print. It has been replaced by the new book, *Common COMAL Reference*. Cartridge COMAL users now can have a quick reference to all the keywords, including those in the built in packages, in a convenient Doc Box style. *Cartridge Keywords* is already a best selling book. Here is the summary of best selling books in the past few months:

December 1987

#1 - Library of Functions & Procedures
by Kevin Quiggle
#2 - COMAL Quick & Utilities #2/#3
by Jesse Knight
#3 - COMAL From A to Z
by Borge Christensen
#4 - Cartridge Graphics & Sound
by Captain COMAL's Friends
#5 - Foundations with COMAL
by John Kelly

January 1988

#1 - COMAL Workbook
by Gordon Shigley
#2 - Library of Functions & Procedures
by Kevin Quiggle
#3 - COMAL Quick & Utilities #2/#3
by Jesse Knight
#4 - Foundations with COMAL
by John Kelly
#5 - Packages Library #2
by various users

February 1988

#1 - COMAL Workbook
by Gordon Shigley
#2 - COMAL From A to Z
by Borge Christensen
#3 - COMAL Cross Reference
by Len Lindsay
#4 - Cartridge Keywords
by various users
#5 - Library of Functions & Procedures
by Kevin Quiggle ■

COMAL Today Issue 22 May 23, 1988

General

Sprites

Sound & Music

Fun & Graphics

2.0 Packages & Programming

Applications

Reference

Editor
Len Lindsay

Assistant
Maria Lindsay

Artwork
Luther Hux

Contributors
Phyrne Bacon
Richard Bain
Peter Burkinshaw
Borge Christensen
Captain COMAL
Gerard Frey
Dawn Hux
Luther Hux
Bill Inhelder

Contributors
Alan Jones
Tom Kuiper
Len Lindsay
JR Mayhew
Jean Nance
Gary Parkin
Joel Rea
David Stidolph
David Warman

COMAL Today is published by COMAL Users Group, U.S.A., Limited, 5501 Groveland Ter, Madison, WI 53716 and welcomes contributions of articles, manuscripts and programs which would be of interest to readers. All manuscripts and articles sent to COMAL Today will be treated as unconditionally assigned for publication and copyright purposes to COMAL Users Group, U.S.A., Limited and is subject to the Editor's unrestricted right to edit and to comment editorially. Programs developed and submitted by authors remain their property, with the exception that COMAL Users Group, U.S.A., Limited reserves the right to reprint the materials, based on that published in COMAL Today, in future publications. There will be no remuneration for any contributed manuscripts, articles or programs. These terms may be varied only upon the prior written agreement of the Editor and COMAL Users Group, U.S.A., Limited. Interested authors should contact the Editor for further information. All articles and programs should be sent to COMAL Users Group, U.S.A., Limited, 5501 Groveland Ter, Madison, WI 53716. Authors of articles, manuscripts and programs warrant that all materials submitted are original materials with full ownership rights resident in said authors. No portion of this magazine may be reproduced in any form without written permission from the publisher. Local Users Groups may reprint material from this issue if credit is given to COMAL Today and the author. Entire contents copyright (c) 1988 COMAL Users Group, U.S.A., Limited. The opinions expressed in contributed articles are not necessarily those of COMAL Users Group, U.S.A., Limited. Although accuracy is a major objective, COMAL Users Group, U.S.A., Limited cannot assume liability for article or program errors.

Our NEW Address is:

COMAL Users Group, U.S.A., Ltd.
5501 Groveland Terrace
Madison, WI 53716
(608) 222-4432

From the Editor's Disk

by Len Lindsay

I got this issue out while we moved our office **twice** and had three holidays (Thanksgiving, Christmas, New Years). Since you asked for them, this issue is packed with program listings.

Today Disk #22 includes top notch programs **plus** a **FREE** Power Driver interpreter (the compiler is still only available in the Power Box). A most significant COMAL 2.0 program is on *Today Disk #22*: **Buffer Editor**. It is a good example of utilizing packages and our pop over system (since it uses packages, it can't be a type in program). It shows how to use the **Windows** and **Buffer** packages together and provides an easy way to edit buffer text (a text editor of sorts). It also shows how easy it is to pop up commands with our pop over system from *COMAL Today #11* (they are easy to use and easy to add to programs). I am surprised that more people do not utilize them. Maybe this program will convince you how valuable they are. A command summary for **Buffer Editor** is on page 26. This program allows you to add over 10K of instructions text to your program without using up your program memory area!

Sound and Music are neglected areas in COMAL programming. This issue explains how to use the **playscore** command in the 2.0 cartridge and includes a program that will create files ready for use with it. Then we have a Music system for both COMAL 0.14 and 2.0. Both are listed in full for your review.

Sprites are great fun. They are also an area of confusion for many beginning programmers. COMAL includes many commands for controlling sprites (no need for POKE or PEEK). We have had many articles and Sprite tutorials in past issues, but Dawn and Luther Hux have an interesting approach with **Walking Sprites** (checkout the back cover to get an idea of their walking boots program).

After the Sprite Tutorial, comes a program that displays Animal Images on the screen as sprites. It works with both COMAL 0.14 and 2.0 (but not Power Driver due to the many sprite images used).

A new improved sprite editor program for COMAL 2.0 is included on *Today Disk #22* and the Command Summary is on page 25.

We use a program called **Davids Directory Designer** to put title boxes into our disk directories. Put one of our disks into your drive and look at its directory to see what I mean. Now we are presenting a **Directory Designer** for COMAL 2.0 users that does the same thing, complete with pop overs! It is on *Today Disk #22* (it uses packages so can't be a type in program) and its command summary is on page 30. And that's not all! Two different people sent us programs to re-organize a disk directory. So we also are including Phyrne Bacons **Reorder Directory** program in this issue, complete with full listing.

COMAL support on QLink continues, and now includes monthly program challenges! See page 5 for the January and February challenges. Plus, now Joel Rea is assisting us with program uploads, so we expect more COMAL 0.14 and 2.0 programs to be on QLink for you to download. Upload your programs there, then send EMail to **COMALite J** advising him to process the files. It is easy to get to our COMAL section on QLink. Just sign on, go to Commodore Information Network (CIN) then:

- Choose Commodore Community
  - Choose Programmers Workshop
    - Choose COMAL

In addition to a program library, there is a Message Board for your COMAL questions and comments. Plus our own conference room where we hold national meetings every First Sunday and Second Thursday at 8pm Eastern Time. ■

COMALites Unite

Life After COMAL

by Richard Bain

As some of you may know, I have had considerable incentive to look at want ads recently. None to my surprise, I have seen a *few* more ads for C programmers than for COMAL programmers. As a result, I decided to take a crash course in C. The following is a biased review of how COMAL compares to the professional language, C. The opinions expressed here are entirely my own, although they are generally consistent with the editors of *COMAL Today*.

The first thing I noticed when I started to learn C was that not one of the 650+ pages of documentation included with the language mentioned whether a printer could by accessed from a C program. COMAL can access the printer with the **SELECT "LP:"** command and normal **PRINT** statements.

The next thing I noticed related to procedures and functions. As a historical note, the original version of COMAL did not have separate structures for procedures and functions. A function was a procedure which returned a value. C is just the opposite. A procedure is a function which does <u>not</u> return a value. Of course, modern versions of COMAL make a clear distinction between the two structures.

I soon tried the IF structure of C. Up until now, I had never truly appreciated the COMAL **IF THEN ELIF ELSE ENDIF** structure. C lacks the **ELIF** portion of this structure. C does allow you to expand **ELIF** into **ELSE IF**, but the result can be very clumsy. **ELSE IF** involves nested levels of **IF** structures whereas **ELIF** involves only one level. Therefore, a C program using several levels of **ELSE IF** will have a large amount of indentation (if you choose to add the spaces yourself) yielding a false sense complexity. Furthermore, the C **IF** structure will end with several '}' characters ('}' is the character C programs use to end a structure). This can easily lead to careless errors placing the correct number of '}' characters in the correct place. These errors are hard to trace. COMAL neatly ends the **IF** structure with the keyword, **ENDIF** (no matter how many optional **ELIF** statements are included within the structure).

The next thing I noticed about C was that it lacks a large variety of miscellaneous commands. These missing commands include floating point routines, and routines to manipulate strings. In defense of C, it comes with a couple hundred standard library routines to provide these and other tasks, but those routines leave something to be desired. For example, C has a library routine similar to the COMAL **IN** statement. In COMAL the result of **"B" IN "ABC"** is **2**. In C, the result would be something like **$FF32**. This represents the address of the character "B" within the string "ABC". It is up to the programmer to determine the address of the start of the string "ABC", the number of bytes between the two addresses, and the number of characters this represents. The C **STR$** function is just as bad. It returns a string of digits without regard for the decimal point. One of the integer parameters of the C **STR$** function is set to indicate where the decimal point belongs.

By this time, I was ready to put down the C manual, and turn on my computer and start running C. I was disappointed (but not surprised) that the editor didn't flag the syntax errors in my early programs. What did surprise me was the time required to compile my short (10-20 line) programs. This sometimes took as much as a minute. COMAL can SCAN a program of several hundred lines in less than a second. When my programs finally started running, I discovered one more thing about C. C lacks an error handling structure.

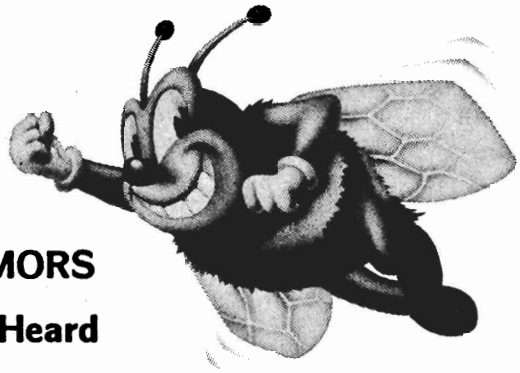
more»

Conclusions:

In the real world, people who want to become professional programmers must learn C. I have gained an appreciation why people who possess the skill and knowledge required to program in C can charge a high price for their services. However, I fail to understand why a businessman would be willing to pay for that particular expertise. Far less experienced programmers can use COMAL to produce quality programs in a fraction of the time required to develop a program in C.

Learning to program in COMAL has still proven valuable, even in the real world. COMAL is the language of choice for all the programming I do for my personal use. I have also been able to apply the structured programming concepts of COMAL directly to C. In fact, I successfully translated my expression analyzer program from *COMAL Today #17* to C within a few days of writing my first lines of C code. I don't believe anyone could learn C as a first language that quickly.

Editors notes: the same day I got this article from Richard, I got EMail on the QLink system from David Skinner who works in the San Francisco area. He is doing professional programming with COMAL. He gradually is convincing small businesses that COMAL is an intelligent programming choice (as Richard has pointed out in his article). However, the $600 price on IBM PC COMAL is significant, though most businesses soon realize that it is worth it for the time it saves. He is COMAL Dave on QLink, and the winner of our February Programming Challenge. You can keep in touch with Richard on QLink as well, his QLink ID is COMALite B. ■



RUMORS We Heard

by Captain "*Buzz*" COMAL

There is a new German company now testing a version of COMAL for the Amiga. It is said to be the fastest COMAL out, even faster than the IBM PC COMAL from UniComal. It is written in Machine Code for speed and will have window and graphic capabilities. They are sending me a test version, but mail from Germany takes a week or more to get here, so I couldn't write about it now. Next issue I should be able to give you more information and details on it. They plan on releasing the final this summer. This is a new company, not related to Mytech. I personally know the main programmer (who has written several final COMAL 2.0 implementations already). He does great work, and I am looking forward to a fantastic Amiga COMAL, but can't say more at this time. I'll probably have to buy an Amiga for us now.

I also talked with UniComal in Denmark. They are working on a LAN version of COMAL for the IBM PC and PS/2 computers (and compatibles). This would be for use in a business environment. They also are developing a **mouse** package for their IBM COMAL. This will be a nice addition. (We already published a **mouse** package for the C64 COMAL cartridge in issue #18.) Finally they said that they also are working on a **matrix** package. This should be very popular with our readers too.

Power Driver has now replaced COMAL 0.14 for the Commodore 64. It is available for download from QLink as a self disolving ARC file (you are welcome to upload this SDA file to other networks). We are including it on *Today Disk #22*, and expect the *UpTime* disk people to include it on one of their disks soon (maybe even a *LoadStar* disk soon). ■

QLink Challenge

Center Challenge

The official January 1988 QLink COMAL Programming Challenge was to write a program that would input up to 38 characters, then print those characters centered on the screen surrounded by ***'s. For example:
Input: Hello there.

```
**************
*Hello there.*
**************
```

SUBJ: Center - entry #1 FROM: Captain C
01/22/88 S#: 3132

```
dim text$ of 38, star$ of 40
for x=1 to 40 do star$(x:x)="*"
input ">": text$
page
pos=(40-len(text$)) div 2
print at 12,pos: star$(len(text$)+2)
print at 13,pos: "*"+text$+"*"
print at 14,pos: star$(len(text$)+2)
```

SUBJ: Center - Entry #2 FROM: Jimmy l
01/23/88 S#: 33958

```
dim star$ of 40, text$ of 38
input "enter text: ": text$
page
for i=1 to len(text$)+2 do star$(i:i):="*"
space:=(40-len(text$)) div 2
for i=1 to 11 do print
print tab(space), star$
print tab(space), "*"+text$+"*"
print tab(space), star$
```

SUBJ: Center - Entry # 3 FROM: Jimmy l
01/24/88 S#: 18627

```
INPUT AT 0,0,38: ">": text$
TIME 0
DIM star$ of 40
PAGE
FOR i=1 to LEN(text$) DO star$(i:i):="*"
space:=(40-LEN(text$)) DIV 2
PRINT AT 12,space:"*"+star$+"*"
PRINT AT 13,space:"*"+text$+"*"
PRINT AT 14,space:"*"+star$+"*"
PRINT TIME/60;"Seconds"
```

SUBJ: Center - ch2a.xojo FROM: Xojo
01/24/88 S#: 22819

```
// delete "ch2a.xojo"//comal 2.0
// save   "ch2a.xojo"
// size  = 353;  jiffies = 7
// score = 41.18 byte*seconds
INPUT "Enter string: "+""13"": a$
TIME 0
PAGE
stars$:="***** . . . (38 of 'em) . *"
long:=LEN(a$); posn:=(39-long)/2
PRINT AT 12,posn: stars$(:long+2)
PRINT AT 13,posn: "*",a$,"*"
PRINT AT 14,posn: stars$(:long+2)
PRINT AT 21,15: TIME
END "COMAL rules!"
```

If I delete all the comment lines and the last line, the program becomes, of course, smaller. It still runs in 7 jiffies, but scores 22.63. This gives some insights. Notice how SIMILAR all the solutions are, so far. Notice that //comments cost very little in storage, and NOTHING in runtime! It's almost boring! It is also counterproductive and it encourages bad style to make the contest minimal size.

SUBJ: entry #4 FROM: Len Lindsay
01/25/88 S#: 97177

```
dim text$ of 38, star$ of 40
input ">": text$
page
pos=(40-len(text$)) div 2
for x=1 to len(text$)+2 do star$(x)="*"
print at 12,pos: star$
print at 13,pos: "*"+text$+"*"
print at 14,pos: star$
```

more»

Note that the FOR loop assignment IS a shortcut substring. IBM COMAL will not allow this shortcut but uses another: **star$(:x:)="*"**

To allow all version to use the same use this: star$(x:x)="*"

SUBJ: Center-Entry #6 FROM: DavidW57
01/31/88 S#: 817069

```
DIM text$ of 38
asterisk$=""
PRINT "Input text"
INPUT "":text$
PRINT chr$(147)
ln=len(text$)+2
for i#=1 to ln do asterisk$(i#):="*"
col=(40-ln)/2+1
PRINT AT 12,col:asterisk$(1:ln)
PRINT AT 13,col:"*"+text$+"*"
PRINT AT 14,col:asterisk$(1:ln)
```

SUBJ: Center -- Entry #7 FROM: COMALite J
01/31/88 S#: 834037

```
INPUT AT 0,0,40: "Text:  ": line$
length#:=len(line$)+2
DIM stars$ OF length#
stars$:="**************************************
******" //42 stars -- Wrap Line
center:=20-length DIV 2
PRINT AT 12,center: stars$
PRINT AT 13,center: "*",line$,"*"
PRINT AT 14,center: stars$
```

Listed for COMAL 2.0. For Power Driver, insert DIM line$ OF 40 at the beginning, and remove the ,40 from the INPUT AT statement. For COMAL 0.14 [after modifying for Power Driver], remove the AT 0,0: from the INPUT statement.

Backwards Challenge

For the official February 1988 QLink COMAL Programming Challenge, write a program to input a line of text (up to 80 characters long including any characters the user might type, such as commas, colons: or "quotes"!), then print the text out BACKWARDS! If INPUT is:

Palindrome: "Madam, I'm Adam!"

the program should output:

"!madA m'I ,madaM" :emordnilaP

The program must store the reversed version of the INPUTted line into a string variable which remains available after the program finishes executing! It is not enough to simply print out the characters of the inputted string in reverse order! We are testing knowledge of string handling, and ways to make it more efficient.

To keep things fair, and to avoid penalizing people for use of the nice, long, descriptive variable names which COMAL permits (with little penalty, since they are stored only once, no matter how many times they are referenced), the length of the COMAL entries will be determined by taking the length of the CODE section only, not counting the Name Table.

Scoring will be the same as in the QLink BASIC challenge: bytes of code * the number of jiffies to execute (not counting time to type in the input string). Lowest score wins.

SUBJ: Backwards Entry #1 FROM: Captain C
02/16/88 S#: 58132

```
DIM t$ OF 80, s$ of 1
INPUT t$
z=LEN(t$)
FOR x=1 TO z DIV 2 DO s$=t$(x);t$(x)=
t$(z-x+1);t$(z-x+1)=s$//wrap line
PRINT t$
```

That is the compact version. Not nice looking, but the BASIC programmers challenge would like it since it is compact. This looks better:

more»

```
DIM text$ of 80, swap$ of 1
PRINT "Enter text:"
INPUT "":text$
long=len(text$)
FOR char#=1 TO long DIV 2
  swap$=text$(char#)
  text$(char#)=text$(long-char#+1)
  text$(long-char#+1)=swap$
ENDFOR char#
PRINT text$
```

Note: this uses a substring shortcut. IBM COMAL needs both start and end positions for substrings (or a different type of shortcut)

```
text$(char#) // Power Driver
text$(char#:char#) // All COMALs
```

SUBJ: Backwards 2 FROM: Jimmy 1
02/18/88 S#: 829843

```
DIM word$ OF 80, backwards$ of 80
INPUT word$(:80)
FOR i#=80 to 1 STEP -1 DO backwards$(81-
i#:81-i#):=word$(i#:i#) //wrap line
PRINT backwards$
```

SUBJ: Challenge Entry 3 FROM: DavidW57
02/21/88 S#: 5616

```
DIM text$ OF 80, reverse$ OF 80
INPUT "": text$
l:=LEN(text$)
FOR i#=l TO 1 STEP -1 DO reverse$(l+1-i#):=
text$(i#) //wrap line
PRINT reverse$
```

SUBJ: Backwards 4 FROM: COMAL Dave
02/18/88 S#: 839395

```
DIM text$ of 80, temp$ of 80 //COMAL 2.0
temp$:=""
INPUT "Enter text: ": text$
FOR char#:=LEN(text$) TO 1 STEP -1 DO temp$
:+text$(char#) //wrap line
PRINT temp$
```

For Power Driver change the FOR loop to:

```
FOR char#:=LEN(text$) TO 1 STEP -1 DO temp$
:=temp$+text$(char#) //wrap line
```

The other solution offered was

```
ZONE 0
DIM text$ OF 80
INPUT "Enter text: ":text$
FOR char#:=LEN(text$) TO 1 STEP -1 DO
PRINT text$(char#) //wrap line
```

Submitted by SBCCC, South Bay, Los Angeles

SUBJ: my challenge entry 5 FROM: Capt zen
02/27/88 S#: 71891

From the looks of the entries so far, it seems I'm taking a different approach. It uses more variables than the other entries I've seen, but the loop only iterates for half the length of the string. Maybe I'll save some time there. It'll also be interesting to see if it's faster to print each byte of the string, or flip the string in memory, and print the whole thing once.

```
dim byte$ of 1
dim line$ of 80
input line$
print
count#:=len(line$)
back'ptr#:=count#
front'ptr:=1
count#:=count# div 2
while count#>0 do
  byte$:=line$(front'ptr#)
  line$(front'ptr#):=line$(back'ptr#)
  line$(back'ptr#):=byte$
  front'ptr#:+1
  back'ptr#:-1
  count#:-1
endwhile
print line$
```

Send in ideas for future Challenges please!

more»

Backwards Challenge Results

COMAL entry times and sizes use Power Driver (with 2.0 times/bytes in parentheses).

In reverse order of size:

9. Capt Zen	(COMAL)	172 (179)	bytes
8. Captain C #2	(COMAL)	145 (155)	bytes
7. Captain C #1	(COMAL)	110 (119)	bytes
6. Jimmy L	(COMAL)	93 (102)	bytes
5. COMAL Dave	(COMAL)	94 (97)	bytes
4. DavidW57	(COMAL)	88 (93)	bytes
3. Grey Ghost #1	(BASIC)	81	bytes
2. Grey Ghost #3	(BASIC)	47	bytes
1. Grey Ghost #2	(BASIC)	34	bytes

Uh oh, it looks bad for COMAL here! All BASIC entries beat the COMAL entries for space!

In reverse order of speed:

9. Grey Ghost #3	(BASIC)	1,757	ms
8. Grey Ghost #1	(BASIC)	1,469	ms
7. Grey Ghost #2	(BASIC)	1,384	ms
6. Jimmy L	(COMAL)	1,274 (586)	ms
5. Capt Zen	(COMAL)	1,101 (475)	ms
4. Captain C #2	(COMAL)	956 (514)	ms
3. Captain C #1	(COMAL)	945 (511)	ms
2. DavidW57	(COMAL)	910 (490)	ms
1. COMAL Dave	(COMAL)	627 (254)	ms

WOW! Note also that while the 2.0 times were an **average** of about double the speed of Power Driver times, this ratio was not absolute. Capt Zen's entry was faster than all others except for COMAL Dave's in 2.0, but not Power Driver.

In reverse order of total score:

9. Capt Zen	(COMAL)	189,372	(85,025)
8. Captain C #2	(COMAL)	138,620	(79,670)
7. Grey Ghost #1	(BASIC)	118,989	
6. Jimmy L	(COMAL)	118,482	(59,712)
5. Captain C #1	(COMAL)	103,950	(60,809)
4. Grey Ghost #3	(BASIC)	82,579	
3. DavidW57	(COMAL)	80,256	(45,570)
2. Grey Ghost #2	(BASIC)	74,736	
1. COMAL Dave	(COMAL)	58,938	(24,638) ■

Questions

Question: When it first became available, I purchased a C64 COMAL 2.0 (tan) cartridge. Now I would like to get Super Chip to take complete advantage of all the capabilities offered by the COMAL 2.0 system, but there is no empty socket in my cartridge. - David Agee

Answer: Super Chip added about 100 commands to the system, so we knew that all the early cartridge users would want this too. So we spent extra time to create a disk loaded Super Chip, appropriately called Superchip On Disk. Just load in the full Super Chip from disk once, and the computer is tricked into thinking there really is a chip in your cartridge! It will work with both the tan and black cartridges.

Question: Which COMAL disks can we put into our clubs disk library that may be copied for our members? I am president of a small Commodore pen pal users group, "Meeting 64/128 Users Through the Mail", and am trying to talk up COMAL in our newsletter. - Jean Nance, St. Joseph, IL

Answer: This has been confusing! Recently we have taken steps to clarify things, adding a line at the bottom of our disk labels that states: Please do not copy or You may copy. Meanwhile, these are disks that may be copied: Today Disks #1-#19, all User Group disks, Shareware disks, Tutorial disk, Auto Run disk, Bricks, Paradise, Sampler. Please do not copy the rest, as disk sales are our main source of income, allowing us to create more new disks, books, and newsletters ... and buy new computers for new COMAL implementations as they are released (like Amiga soon). Also, free to copy does not include selling! We can make special arrangements for any person, club, school, or company to sell our disks. Any other ideas? Please let me know. NOTE: the new Power Driver Tutorial may be copied, and includes the FULL Power Driver interpreter system. This is a good starter disk for newcomers to COMAL. ■

Letters

SMU Support

Dear Len, I have *COMAL Today* from issue #1. The reason that I am a COMAL supporter is that a professor at S.M.U. (Southeastern Massachusetts University) said to me during a symphony orchestra rehearsal, "Roger, you should try COMAL".

Previously, the Captain COMAL bit had turned me off, since it *appeared* that the language was aimed at kids --- not a serious language. My sincere hope is that the low has been reached, and that membership will, indeed, rise to and beyond the 100,000 mark! - Roger Walen

Recovery

Dear Sir: I would like to note that the disk recovery program on *Today Disk #20* saved me more than $20. My grandson, bit a commercial disk. I tried to recover by using Fast Hackem, but the error was still present. I used the program on *Today Disk #20* on the disk made by Fast Hackem and was able to recover the bad sections. I enjoy your publication very much. - John Hieber, Oak Lawn, IL

From Down Under

Dear Len, It has long been a mystery to me why COMAL is not the language, and why more people are not subscribing to your excellent group. I have never regretted one penny that I have spend over the last couple of years buying from the User Group, and remember that everything is at least three times the cost to me than anyone in the USA. I can understand people wanting COMAL for the Amiga, but it will come. I am sorry that I can't do anything tangible to help, but please be assured I support you in your efforts for COMAL. Best wishes... Mark Sims, New Zealand

For Sale

Dear Folks, I am very happy to have been exposed to COMAL and indeed to have been deeply interested in it. However, now my interest is almost exclusively with IBM PC. In this area, the pricing for COMAL has lead me elsewhere. [*Sorry, but we can't control the IBM COMAL price, it is imported from Denmark, and is probably the fastest and best COMAL available today*]. The following materials are offered for sale: [Clifton M Winebert, RD 6, Box 492, PUnxsutawney, PA 15767]

```
COMAL 2.0 tan/beige cartridge - $20
COMAL 2.0 black cartridge & Quick Chip - $50
SuperChip - $10

Documentation Package - $20 includes: COMAL
Handbook, COMAL Today #1-#16, COMAL From
A To Z, COMAL 2.0 Packages, Cartridge
Graphics and Sound, and 14 disks
```

IBM COMAL Note

Dear Len, I recently transferred into a department here at Union Camp that uses IBM-alikes extensively. We have 6 on a LAN plus several stand-alones. There are no department standards on application software. Each individual uses whichever package or language that suits his fancy. I have suggested to my boss that this has short-term problems which only come to light when anyone (including the application developer) attempts to modify or upgrade the application. I evaluated several languages in a spreadsheet format and reached the conclusion (as I knew I would) that COMAL was the best choice for a department standard - except for the fact that it is not well known.

I do not know what the answer is. I have preached COMAL with an evangelical fervor for three years. The school system wouldn't even listen and has since decided to put the C64s in a closet and buy Apples (and the school system

more»

can't even afford field trips to the local fire station, they say!)

I personally have decided to keep my C64 a while longer to enjoy the COMAL environment. But the C64 will not last forever, and an upgrade is surely in my future. It's hard to justify a $400 COMAL when Turbo Pascal can be had for $49 (even though I know the TP environment is crude and the language is picky). Perhaps I just don't know how good the IBM COMAL environment is.

Well, I am sure this letter hasn't brightened your day, but I did want you to know that there are some of us who have shared your vision and hope that provides you with some moral support. I wish you the very best in the future and I sincerely hope our dream does not die. - Bruce Powell, Franklin, VA

[*We import IBM COMAL and can't control its price. It is the* **best** *COMAL we have now!*]

Scientific Uses

Dear Mr. Lindsay: I use COMAL exclusively in my scientific work. I think it is the program that suits best my needs, and the information in your publication and on the *Today Disks* have been invaluable to me.

I have used computers as part of my work for about 30 years, although I was never a full-time computer operator. My profession is analytical chemistry. About 3 years ago, Craig Van Degrift introduced me to COMAL. I soon got COMAL 2.0. This is what I use since, both on my C64 at home, and the IBM PC at work. Believe me, I am sold on COMAL, not only for its logic, but mainly for the easy disk-handling, error trapping, debugging, tracing, files, etc. Every month, in a Commodore journal, I see announcements of "revolutionary features" added to BASIC which I had used for 2 years in COMAL already. - Kurt Heinrich, Rockville, MD

School

Dear Sirs: As a high school computer science teacher, I became interested in COMAL a few years ago when I first heard about it. I have been a subscriber of *COMAL Today* since the first issue. It took several years to convince our school system to use COMAL as an alternate to BASIC, but we succeeded. This is now the second year in which we use COMAL to introduce our students to computer programming. By the end of this year, some 500 students in our high school will have learned to program in COMAL. - Ronald France, Cuyahoga Falls, OH ■

New York City and Washington DC

Message From Captain COMAL & Len Lindsay: Any COMALites in these areas are welcome to stop by and say hello. I will be at the BUG 64 meeting June 16 (Bronx 64 Users Group, PO Box 523, Bronx, NY 10475) and a special meeting in the DC area on June 18 (Bob Bain, 10200 Leslie St, Silver Spring, MD 20902). I haven't had time to attend Commodore shows recently, and will enjoy these two meetings. Maybe even one in Boston on June 13 or 14 if it can be arranged (let me know out there). ■

2.0 convert IMAG. to SHAP. files

Use this program to convert a sprite image file from IMAG. format into SHAP. format required for use with the **loadshape** command.

```
DIM image$ of 64
USE sprites
INPUT "Filename? imag.": name$
OPEN FILE 2,"imag."+name$,READ
READ FILE 2: image$
CLOSE FILE 2
define(1,image$)
saveshape(1,"shap."+name$) ■
```

Walking Sprites

by Dawn and Luther Hux

Have you ever wished you could create and animate your own sprite ideas. This article attempts to teach you what you need to know to produce animated sprites using COMAL.

The code listed here is the code used to move the walking boots in the word game, **shredder,** on *Today Disk #20.* The code to run the boots independent of the game is on *Today Disk #22.* **LOAD "walking boots"** and **RUN** it.

The unique feature of this article is the use of a frame by frame guide to assist in writing the algorithm for the complex animation of two sprites in rhythm. While the process sounds difficult, the instruction will be kept on introductory level. It starts with writing sprite data using a design sheet through the steps on how to put the boots in motion. This material covers some instruction found in earlier tutorials. I encourage you to read any articles available to you but this article does not depend on any previous sprite knowledge.

The complexity mentioned here refers to changing sprite shapes and positions in a special rhythm and not to the fact that there is more than one sprite on the screen.

First things first. What is a sprite? A sprite is a flexible image who's shape and color can be custom designed by the programmer and its position and motion can be controlled by the user or programmer. Most users relate to sprites as alien space ships and asteroids. Sprites make animation simpler since defining an entire screen for each image movement would be slow and memory consuming. A sprite takes very little memory and responds quickly to instructions for shape or color changes. They also have the ability to detect collision with other sprites, used as targets, or with locations on the screen.

COMAL makes the color, position, expansion and shape assignment very simple with literal keywords eliminating an endless list of POKEs. Also COMAL does not require you to specify sprite shape memory locations and pointers. COMAL also operates sprites at any position on the screen without resetting the position markers when they exceed 255. That simplifies making the boots walk completely across the screen when the screen is 319 pixels across.

The challenge is deciding what the sprites look like and how they are to move. Of course, this applies to using any language. At least COMAL lets you get on with the process with ease once the artwork and algorithm is designed.

All languages set up sprite shapes in much the same way. **Figure 1** (on the cover) shows a completed sprite worksheet. A blank worksheet, which you can copy, is at the end of this article. Each square represents a pixel on the screen that can be turned on or off for hi-resolution, single color sprites. Multi-color sprites are designed differently and are not covered in this article.

For the boot design the first problem is to decide exactly what a boot looks like at each phase of walking. I choose only three images, heel, flat and toe shown in **figure 2** (on the cover). The boots are then sketched on the worksheet. The outline shows which pixels are to be turned on. Of course, the artwork looks very much like needle-point when finished.

Once the artwork is acceptable in appearance you must write the sum of each horizontal row, in each section, in the subtotal column for that section. This is done by adding the column values, assigned at the top of the chart, of each pixel that will be turned on. Pixels that remain off have 0 value. To turn on the pixel to the far right the data value is 1. The far left pixel has the value of 128. To turn on the entire row the sum of all the column values is

more»

255. The artwork is divided into three sections labeled A, B, and C. The maximum total of one horizontal row in each of the three sections is 255. To present the data for an entire row (all three sections, A, B, and C) you need three numbers in the three subtotal columns on the right side. This is repeated for all 21 horizontal lines of the sprite so the completed sprite data has 63 number entries separated by commas. To type the numbers into the data lines, read them from the chart the same as reading a book, left to right starting from the upper left and going down the lines. It usually takes three DATA lines to enter the 63 entries for one sprite. See **lines 250 to 270** below. There are three shapes in this design so there are three different groups of data. To get the shape data into sprite memory you must **READ** and **DEFINE** the data. Each shape has a different shape number so we can call the shapes as needed. In BASIC you have to define the storage area and a vector (pointer) for each data group. In COMAL this is done for you. Simply use a loader such as the one shown in **lines 180 to 240** below and the shape will be stored in a reserved location specified by the number following the keyword **DEFINE. Line 240** defines this data group to shape 2 in this example.

We will use shape 2, which describes a boot flat on the floor, for line by line analysis. Remember we are working with three shapes now. Later we will assign them to two sprites in various patterns. Each sprite has access to the same or different shapes.

Line 180 dimensions the string **image2$.** Image just a description of what the variable is being used for. The 2 refers to the second image and the $ is the symbol for a string variable.

Line 190 is a FOR/NEXT loop to read in the 63 data numbers that describe the shape.

Line 200 reads the data into the variable named **boot'flat.** It is not a keyword but a variable that describes what the shape looks like for easy recognition.

Line 210 accumulates the data read into the FOR/NEXT loop variable, **boot'flat,** into the variable string, **image2$.**

Line 220 closes the FOR/NEXT loop. Notice that we dimensioned for 64 bytes and read only 63.

Line 230 adds the 64th byte of data. 0 means a standard hi-res image. 1 means multi-color.

Line 240 places **image2$** into the memory location for shape number 2. The first shape in the completed program (heel) was defined as number 1 and the third shape (toe) was defined as shape number 3. The remaining lines in this group of code are the DATA lines. Here is an example of how to read shape number2:

```
0180 DIM image2$ of 64
0190 FOR x:=1 TO 63 do
0200  READ boot'flat
0210  image2$:=image2$+CHR$(boot'flat)
0220 ENDFOR x
0230 image2$:=image2$+CHR$(0)
0240 define 2,image2$
0250 DATA 0,0,0,63,224,0,63,224,0,63,224,0,
     63,224,0,63,224,0,63,192,0//wrap line
0260 DATA 63,192,0,63,192,0,63,128,0,31,128,
     0,31,128,0,31,128,0,31,192,0//wrap line
0270 DATA 31,224,0,63,240,0,63,248,0,63,252,
     0,31,255,128,28,127,192,28,63,128//wrap
```

Once the data for all shapes are in memory you must setup the screen for sprites by turning on the graphic screen and describing which shape, color, size and screen position is desired. COMAL provides English-like commands for this.

Line 380 calls for a hi-res graphic screen (change the 0 to 1 for multi-color graphics).

Line 390 makes the drawing symbol invisible. This delta shaped drawing marker (called a

more»

turtle) shows the PEN location and direction but we are not drawing in this program. Since it comes with the graphic screen you must turn it off if you are not using the TURTLE.

Line 400 tells the computer that sprite 1 is not to be expanded either vertically or horizontally by placing a 0 in each argument position. To double the width, you can replace the center number with a 1. To double the height, replace the last digit with a 1.

Line 410 repeats the same for sprite 2.

Line 420 sets the color of sprite 1 to white, (color number 1). **Line 430** sets sprite 2 to grey, (color number 12). These are standard Commodore color numbers. The grey color helps the boots in the background appear further away and in the shadow of the white boot. That's an artistic opinion and not something you need to learn.

Line 440 positions sprite 1 at 20 pixels from the left border and 50 pixels from the bottom border. The 0,0 position for COMAL is in the lower left corner.

Line 450 sets sprite 2 against the left border (0) and at the same vertical level as sprite 1. The horizontal movement is listed first and the vertical movement listed second. The center of the screen is 160,99.

Line 460 and **470** are listed last in this batch because they turn the sprites on and describe the shape to be assigned each sprite. You should not **IDENTIFY** a sprite until you have completely described size, color and position to prevent the sprite from flickering in another position or color left over from a previous presentation. The first number is the sprite and the second number the shape you wish to assign to that sprite. You can re-assign different shapes in a sequence to create animation. This program hardcodes the sequence but in simple animation you can replace the shape number with a variable and cycle the shapes by resetting the value of the variable with the shapes you wish to identify.

The following code places both boots flat on the floor as shown in **figure 3** (on the cover). This position is maintained in the Shredder program while other action is taking place.

Line 480 sets the color for **line 490** which uses **PLOTTEXT** to print a line across the screen.

Line 495 in this program imitates this with a delay loop to hold the boots still for a few seconds before the walking routine takes over.

```
0380 SETGRAPHIC 0
0390 HIDETURTLE
0400 SPRITESIZE 1,0,0
0410 SPRITESIZE 2,0,0
0420 SPRITECOLOR 1,1
0430 SPRITECOLOR 2,12
0440 SPRITEPOS 1,20,50
0450 SPRITEPOS 2,0,50
0460 IDENTIFY 1,2
0470 IDENTIFY 2,2
0480 PENCOLOR 0
0490 PLOTTEXT 0,25 "------------------------
     ----------------------"//wrap line
0495 DELAY(1500)
```

Sprite 1 is the right boot. Sprite 2 as the left boot has a lesser priority so it will pass behind sprite 1 (lower numbered sprites have priority). This creates a realistic impression of depth.

These boots were made for walking, so we need a loop that imitates the walking motion. At first I couldn't figure how to start or where the loop would come full circle. Using an idea from cartoonist I made up a story board to illustrate the steps frame by frame. Once the artwork shown in **figure 4** (back cover) was complete, writing the code was easier. Beginning with the set-up code that placed the two boots

more»

on the screen, the shape and position of each frame was recorded in the right column.

Since the boot is about 20 pixels long a step distance of 20 pixels was selected and assigned to each column of the drawing. The pictures quickly showed that one boot stayed still for five frames while the other boot went through its paces. The drawing showed that the opposite boot was either a -20, even with, or +20 pixels with the other boot. The variable of the loop is named **walk**; so you will see **walk-20**, **walk** and **walk+20** throughout the loop.

Once all the code was written, the duplicate code lines were eliminated as many of the steps require no change in either position or shape. The code at the beginning appears to be a repeat of the code used to place the boots on the screen the first time, but that is true only for the first pass of the loop, so it is required for the remainder of the loops operations as the boots move away from the left border in 40 pixel increments. Also the earlier code placed both boot flat and that never happens in the walking code so it is not an exact repeat.

When the initial program was operational, I noticed that the right boot rocked back to the heel position before stepping forward. This was observed by using a large number in the delay proc. The boots are initially setup with both boots flat on the floor. The code, written exactly as seen in the drawing, was correct for the illusion of walking but the rocking back of sprite 1 was unnatural. A unique modification had to be made to eliminate this one motion at the start of the walking routine so I moved the code labeled transposed to 700 to the end of the loop. That way it is not processed at the initial entry into the loop but it is still there for the following passes. Since there is no delay between the end and beginning of the loop there is no difference in the performance of the remaining passes. The loop is a circle and you can jump into it at any point.

When sprite number 1 reaches the 60 column in **figure 4**, the value of **walk** has been set to 60 on the second pass of the loop. At this point frame 1A and 1B are the same except that the value of **walk** has been increased from 20 to 60. Sprite number 2 is instructed to position itself at **walk-20** which equals 40 and that is where it is located on the chart in frame 1B. Here's the code from the chart.

```
0500 //******* boots walk *******
0510 FOR walk:=20 to 300 STEP 40 DO
0520  SPRITEPOS 1,walk,50
0530  //IDENTIFY 1,1 (transposed below to 700)
0540  SPRITEPOS 2,walk-20,50
0550  IDENTIFY 2,2
0560  delay(100)
0570  IDENTIFY 1,2
0580  IDENTIFY 2,3
0590  delay(100)
0600  SPRITEPOS 2,walk,50
0610  delay(100)
0620  SPRITEPOS 2,walk+20,50
0630  IDENTIFY 2,1
0640  delay(100)
0650  IDENTIFY 1,3
0660  IDENTIFY 2,2
0670  delay(100)
0680  SPRITEPOS 1,walk+20,50
0690  delay(100)
0700  IDENTIFY 1,1 //(transposed
0710  //   from above, line 530)
0720 ENDFOR walk
```

Notice that the sprites can walk right off the right side of the screen without additional work to get to positions past 255 on the screen.

When the animation is over you should hide the sprites while you reposition them. Otherwise you will get a flickering image as the sprite jumps from the last position to the reset position. To reactivate the sprite simply **IDENTIFY** it and assign a shape number.

more»

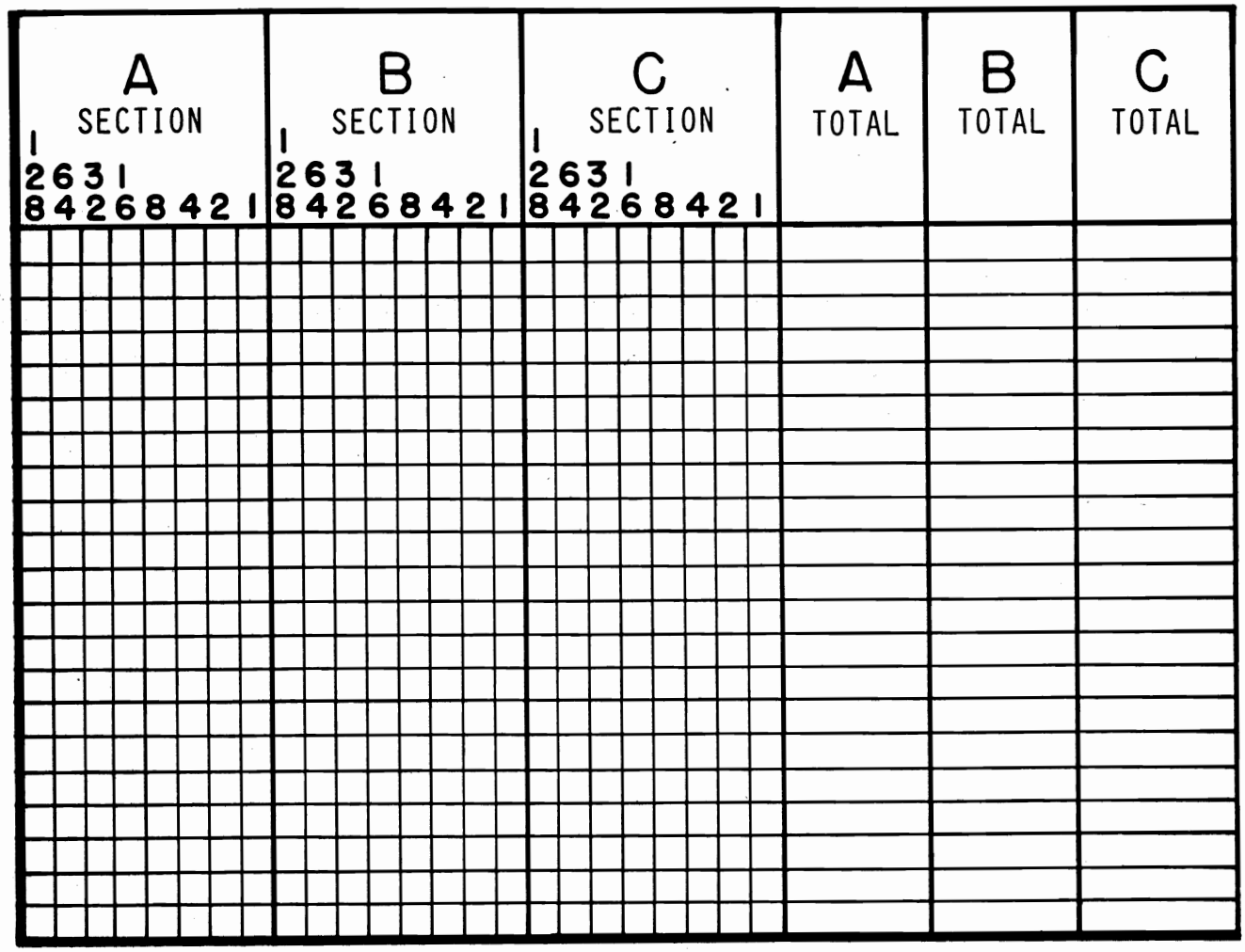
Line 780 returns the program to the text screen. You can also return to the text screen by pressing «F1» (function key 1) after the program stops. To get back to the graphic screen press «F2».

```
0740 HIDESPRITE 1
0750 HIDESPRITE 2
0760 SPRITEPOS 1,0,50
0770 SPRITEPOS 2,20,50
0780 SETTEXT
```

The following delay procedure was used to eliminate repeated FOR/NEXT DO NULL loops within the boots'move proc. It was called by using the word delay followed by the amount of delay desired in ()'s as shown in **line 560.**

```
0810 PROC delay(length)
0820   FOR d:=1 TO length DO NULL
0830 ENDPROC delay
```

That covers the code for making the boots move. I hope there is enough explanation to help you install and animate your own designs.



Animal Alphabet

by Jean Nance

Animal Alphabet is an educational program for children. Younger children may choose to enter any letter of the alphabet to see an animal whose name begins with that letter. Older youngsters may choose to type the entire name to see the animal. The program was written at the request of a friend who has two small children. She wanted something that would give immediate feedback to the baby when a key was pressed, but also be entertaining and educational for the older youngster who is learning to read.

There are versions for 0.14 and 2.0. In the 0.14 version, a sprite file is loaded at the beginning of the program, and only 8 sprites can be shown at a time (since it uses more than 16 sprite images, it cannot be run under Power Driver). In the 2.0 version, the sprites are linked to the program (on *Today Disk #22*) so they do not need to be loaded. All 26 sprites can be shown on the screen at one time, using the **stampsprite** command.

The sprites were designed with sprite'designer2, available on *Today Disk #12*, the *Sprite Disk*, or *Utility Disk #1*. The sprites are in monocolor, since this gives better resolution and makes it easier to do a realistic animal shape. Save (S) from the sprite editor saves the shape in a sequential file which is a 64 character string. The string contains information as to which of the pixels on the 24 by 21 pixel sprite are on or off, and whether the sprite is mono or multicolor.

For the 2.0 version, the sprites were loaded into the program, linked to the program, and then the load procedure was erased. For the 0.14 version, the sprites were appended one to another, using the "append" option in the sprite designer program. This file, "*imag.26animals*", is loaded by the program and the sprite definitions assigned image numbers 0 to 25.

For the option of entering a single letter, the key pressed is read with procedure **get'char**, and a 26 item case statement selects the proper sprite image and assigns it a image number, size, shape, color, and position. In the option of typing the entire name, the name is input, the correct text string is assigned its initial, and the same case structure displays the sprite. In either case, if an incorrect answer is given, nothing happens. The parameters of the **show'sprite** procedure are sprite number: **s'num**, imagenumber: **i'num**, width: **wi**, height: **ht**, color: **co**, position on screen: **posx**, **posy**, and position for the name to be printed below the sprite: **y**. (The location of the name will be posx,y).

If you do not have the programs on disk, you may design your own sprite shapes. Use the append option to create a combined file: append "bear" to "ant", then append "cat" to "ant" and so on through the alphabet to "zebra". "Ant" will then be a 7 block file, whose name can be changed to "*imag.26animals*". Or, type in and run the Sprite Image Maker program, which reads sprite shapes from data statements and writes the disk files.

I didn't have a book with an animal alphabet; alternate and perhaps better choices are available. I wanted common animals with short names, that have distinctive and easily drawn shapes. My friend, Diana Diehl-Dryan, who suggested the program, suggested Xenopus, a clawed frog that is available in pet stores and might be known to many children. "Urchin" is of course a sea urchin. I didn't want to include the imaginary unicorn in an educational program along with 25 real animals.

Experiment with colors and shapes, by changing the parameters in the calls to showsprite. You may notice that while most of the sprites are either single width and height, (false,false) or double size and width (true,true) two animals, the newt and iguana, are double width, single height. This made for an elongated animal.

more»

Change the parameters to true,true, to see the proportions of the original design. Other animal shapes could be designed, or the program could be easily adapted to make another alphabet programs, toys, household items, etc.

Animal Alphabet 0.14 Program

```
// do NOT use with Power Driver
dim init$ of 8
dim animal'name$ of 8
dim mobs$(0:25) of 64
open file 2,"imag.26animals",read
for imag'num:=0 to 25 do
 read file 2: mobs$(imag'num)
 define imag'num,mobs$(imag'num)
endfor imag'num
close file 2
intro
setup
//
proc intro
 repeat
  print chr$(147) // page
  print "do you want to see an animal:"
  print
  print " when you type it's initial ? (1) "
  print
  print " or when you type it's name?  (2)"
  print
  input "your choice (1 or 2): ": choice
 until choice=1 or choice=2
endproc intro
//
proc setup
 setgraphic 0
 background 0
 border 0
 pencolor 3
 hideturtle
 clear
 if choice=1 then
  plottext 5,25,"hit any key to see an animal"
  repeat
   get'char(init$)
   show'animal
  until true=false
 else
  text2
  repeat
   print chr$(147) // page
   print "press <f5> for pictures"
   input "type the animal name: ": animal'name$
   setgraphic
   if animal'name$="ant" then init$:="a"
   if animal'name$="bear" then init$:="b"
   if animal'name$="cat" then init$:="c"
   if animal'name$="dog" then init$:="d"
   if animal'name$="elk" then init$:="e"
   if animal'name$="fox" then init$:="f"
   if animal'name$="goat" then init$:="g"
   if animal'name$="horse" then init$:="h"
   if animal'name$="iguana" then init$:="i"
   if animal'name$="jackass" then init$:="j"
   if animal'name$="kangaroo" then init$:="k"
   if animal'name$="lion" then init$:="l"
   if animal'name$="monkey" then init$:="m"
   if animal'name$="newt" then init$:="n"
   if animal'name$="octopus" then init$:="o"
   if animal'name$="pig" then init$:="p"
   if animal'name$="quail" then init$:="q"
   if animal'name$="rooster" then init$:="r"
   if animal'name$="squirrel" then init$:="s"
   if animal'name$="turkey" then init$:="t"
   if animal'name$="urchin" then init$:="u"
   if animal'name$="vulture" then init$:="v"
   if animal'name$="whale" then init$:="w"
   if animal'name$="xenopus" then init$:="x"
   if animal'name$="yak" then init$:="y"
   if animal'name$="zebra" then init$:="z"
   show'animal
  until true=false
 endif
endproc setup
//
proc show'animal
 case init$ of
 when "a"
  show'sprite(0,0,false,false,9,15,170,137,"ant")
 when "b"
  show'sprite(1,1,true,true,9,80,185,137,"bear")
 when "c"
```

more»

```
 show'sprite(2,2,false,false,2,155,170,137,"cat")
when "d"
 show'sprite(3,3,false,false,11,230,170,137,"dog")
when "e"
 show'sprite(4,4,true,true,9,15,130,82,"elk")
when "f"
 show'sprite(5,5,false,false,2,80,115,82,"fox")
when "g"
 show'sprite(6,6,false,false,1,155,115,82,"goat")
when "h"
 show'sprite(7,7,true,true,1,230,130,82,"horse")
when "i"
 show'sprite(0,8,true,false,5,15,170,137,"iguana")
when "j"
 show'sprite(1,9,true,true,9,80,185,137,"jackass")
when "k"
 show'sprite(2,10,true,true,12,155,185,137,
 "kangaroo")//wrap line
when "l"
 show'sprite(3,11,true,true,7,230,185,137,"lion")
when "m"
 show'sprite(4,12,false,false,9,15,115,82,
 "monkey")//wrap line
when "n"
 show'sprite(5,13,true,false,2,80,115,82,"newt")
when "o"
 show'sprite(6,14,false,false,8,155,115,82,
 "octopus")//wrap line
when "p"
 show'sprite(7,15,false,false,10,230,115,82,"pig")
when "q"
 show'sprite(0,16,false,false,9,15,170,137,"quail")
when "r"
 show'sprite(1,17,false,false,2,80,170,137,
 "rooster")//wrap line
when "s"
 show'sprite(2,18,false,false,12,155,170,137,
 "squirrel")//wrap line
when "t"
 show'sprite(3,19,false,false,2,230,170,137,
 "turkey")//wrap line
when "u"
 show'sprite(4,20,true,false,8,15,110,82,"urchin")
when "v"
 show'sprite(5,21,false,false,9,80,115,82,
 "vulture")//wrap line
 when "w"
  show'sprite(6,22,true,true,11,155,130,82,"whale")
 when "x"
  show'sprite(7,23,false,false,15,230,115,82,
  "xenopus")//wrap line
 when "y"
  show'sprite(0,24,true,true,9,15,185,137,"yak")
 when "z"
  show'sprite(1,25,true,true,1,80,185,137,"zebra")
 otherwise
  setup
 endcase
endproc show'animal
//
proc get'char(ref code$)
 code$:=chr$(0)
 while code$=chr$(0) do code$:=key$
endproc get'char
//
proc show'sprite(s'num,i'num,wi,ht,co,posx,posy,
y,a$) //wrap line
 identify s'num,i'num
 spritesize s'num,wi,ht
 spritecolor s'num,co
 spritepos s'num,posx,posy
 plottext posx,y,"        "
 plottext posx,y,a$
endproc show'sprite
//
proc text2
 plottext 0,50,"to see an animal press <f1>,"
 plottext 0,40,"type the name and hit return"
 pencolor 1
 plottext 0,30,"ant,bear,cat,dog,fox,goat,horse,
 iguana"//wrap line
 plottext 0,20,"jackass,kangaroo,lion,monkey,
 newt,pig"//wrap line
 plottext 0,10,"octopus,quail,rooster,squirrel,
 turkey"//wrap line
 plottext 0,0,"urchin,vulture,whale, xenopus,yak,
 zebra"//wrap line
 pencolor 3
endproc text2
```

more»

Animal Alphabet 2.0 Program

```
USE sprites
USE graphics
DIM animal'name$ OF 8
DIM a$ OF 8
DIM init$ OF 1
intro
graphicscreen(0)
background(0)
border(0)
pencolor(3)
hideturtle
clear
IF choice=1 THEN
  LOOP
    plottext(2,0,"type letter")
    get'char(init$)
    choose'animal
  ENDLOOP
ELSE
  LOOP
    plottext(2,0,"hit <RETURN> now")
    WHILE KEY$<>CHR$(13) DO NULL
    PAGE
    textscreen
    INPUT "type animal name: ": animal'name$
    fullscreen
    IF animal'name$="ant" THEN init$:="a"
    IF animal'name$="bear" THEN init$:="b"
    IF animal'name$="cat" THEN init$:="c"
    IF animal'name$="dog" THEN init$:="d"
    IF animal'name$="elk" THEN init$:="e"
    IF animal'name$="fox" THEN init$:="f"
    IF animal'name$="goat" THEN init$:="g"
    IF animal'name$="horse" THEN init$:="h"
    IF animal'name$="iguana" THEN init$:="i"
    IF animal'name$="jackass" THEN init$:="j"
    IF animal'name$="kangaroo" THEN init$:="k"
    IF animal'name$="lion" THEN init$:="l"
    IF animal'name$="monkey" THEN init$:="m"
    IF animal'name$="newt" THEN init$:="n"
    IF animal'name$="octopus" THEN init$:="o"
    IF animal'name$="pig" THEN init$:="p"
    IF animal'name$="quail" THEN init$:="q"
    IF animal'name$="rooster" THEN init$:="r"
    IF animal'name$="squirrel" THEN init$:="s"
    IF animal'name$="turkey" THEN init$:="t"
    IF animal'name$="urchin" THEN init$:="u"
    IF animal'name$="vulture" THEN init$:="v"
    IF animal'name$="whale" THEN init$:="w"
    IF animal'name$="xenopus" THEN init$:="x"
    IF animal'name$="yak" THEN init$:="y"
    IF animal'name$="zebra" THEN init$:="z"
    choose'animal
  ENDLOOP
ENDIF
//
PROC intro
  REPEAT
    PAGE
    PRINT "do you want to see an animal:"
    PRINT
    PRINT "when you type it's initial?  (1)"
    PRINT
    PRINT "or when you type it's name?  (2)"
    PRINT
    INPUT "your choice (1 or 2): ": choice
  UNTIL choice=1 OR choice=2
ENDPROC intro
//
PROC choose'animal
  CASE init$ OF
  WHEN "a"
    show'sprite(0,0,FALSE,FALSE,
    9,10,190,160,"ant")//wrap line
    stampsprite(0)
  WHEN "b"
    show'sprite(1,1,TRUE,TRUE,
    9,40,199,150,"bear")//wrap line
    stampsprite(1)
  WHEN "c"
    show'sprite(2,2,FALSE,FALSE,
    2,100,190,160,"cat")//wrap line
    stampsprite(2)
  WHEN "d"
    show'sprite(3,3,FALSE,FALSE,
    11,130,190,160,"dog")//wrap line
    stampsprite(3)
  WHEN "e"
    show'sprite(4,4,TRUE,TRUE,
    9,160,195,145,"elk")//wrap line
```

more»

```
    stampsprite(4)
  WHEN "f"
    show'sprite(5,5,FALSE,FALSE,
    2,220,190,160,"fox")//wrap line
    stampsprite(5)
  WHEN "g"
    show'sprite(6,6,FALSE,FALSE,
    1,255,190,155,"goat")//wrap line
    stampsprite(6)
  WHEN "h"
    show'sprite(7,7,TRUE,TRUE,
    1,10,150,100,"horse")//wrap line
    stampsprite(7)
  WHEN "i"
    show'sprite(0,8,TRUE,FALSE,
    5,85,160,130,"iguana")//wrap line
    stampsprite(0)
  WHEN "j"
    show'sprite(1,9,TRUE,TRUE,
    9,130,140,90,"jackass")//wrap line
    stampsprite(1)
  WHEN "k"
    show'sprite(2,10,TRUE,TRUE,
    12,185,145,95,"kangaroo")//wrap line
    stampsprite(2)
  WHEN "l"
    show'sprite(3,11,TRUE,TRUE,
    7,255,140,95,"lion")//wrap line
    stampsprite(3)
  WHEN "m"
    show'sprite(4,12,FALSE,FALSE,
    9,10,88,58,"monkey")//wrap line
    stampsprite(4)
  WHEN "n"
    show'sprite(5,13,TRUE,FALSE,
    2,70,130,100,"newt")//wrap line
    stampsprite(5)
  WHEN "o"
    show'sprite(6,14,FALSE,FALSE,
    8,65,90,55,"octopus")//wrap line
    stampsprite(6)
  WHEN "p"
    show'sprite(7,15,FALSE,FALSE,
    10,110,90,65,"pig")//wrap line
    stampsprite(7)
  WHEN "q"
    show'sprite(0,16,FALSE,FALSE,
    9,145,90,65,"quail")//wrap line
    stampsprite(0)
  WHEN "r"
    show'sprite(1,17,FALSE,FALSE,
    2,188,73,45,"rooster")//wrap line
    stampsprite(1)
  WHEN "s"
    show'sprite(2,18,FALSE,FALSE,
    12,220,85,55,"squirrel")//wrap line
    stampsprite(2)
  WHEN "t"
    show'sprite(3,19,FALSE,FALSE,
    2,265,90,65,"turkey")//wrap line
    stampsprite(3)
  WHEN "u"
    show'sprite(4,20,TRUE,FALSE,
    8,8,55,25,"urchin")//wrap line
    stampsprite(4)
  WHEN "v"
    show'sprite(5,21,FALSE,FALSE,
    9,65,50,17,"vulture")//wrap line
    stampsprite(5)
  WHEN "w"
    show'sprite(6,22,TRUE,TRUE,
    11,134,25,15,"whale")//wrap line
    stampsprite(6)
  WHEN "x"
    show'sprite(7,23,FALSE,FALSE,
    15,140,58,30,"xenopus")//wrap line
    stampsprite(7)
  WHEN "y"
    show'sprite(0,24,TRUE,TRUE,
    9,202,40,0,"yak")//wrap line
    stampsprite(0)
  WHEN "z"
    show'sprite(1,25,TRUE,TRUE,
    1,265,50,0,"zebra")//wrap line
    stampsprite(1)
  OTHERWISE
    null
  ENDCASE
ENDPROC choose'animal
//
PROC get'char(REF code$)
  code$:=CHR$(0)
```

more»

```
  WHILE code$=CHR$(0) DO code$:=KEY$
ENDPROC get'char
//
PROC show'sprite(spr,imag,wi,ht,
co,posx,posy,y,a$)//wrap line
  identify(spr,imag)
  spritesize(spr,wi,ht)
  spritecolor(spr,co)
  spritepos(spr,posx,posy)
  showsprite(spr)
  plottext((posx-8),y,a$)
ENDPROC show'sprite
```

Creator Maker

(from imag. files into data statements)

This is the program that was used to read the sprite image files from disk, and create a program with data lines that would recreate those files for those typing in the programs.

```
init
write'main'program
write'define'proc
write'write'files
write'sprite'data'lines
PRINT "closing file ...";outfile$
CLOSE
// IMAG.xxx files on this disk:
DATA "ant","bear","cat","dog"
DATA "elk","fox","goat","horse"
DATA "iguana","jackass","kangaroo"
DATA "lion","monkey","newt"
DATA "octopus","pig","quail"
DATA "rooster","squirrel","turkey"
DATA "urchin","vulture","whale"
DATA "xenopus","yak","zebra"
//
PROC init
  DIM infile$ OF 20, outfile$ OF 20
  DIM image$ OF 64
  PAGE
  PRINT "create a program with data lines for"
  PRINT "26 sprite images. working..."
  outfile$:="0:image'lines.lst"
  PRINT "deleting old file ...";outfile$
  DELETE outfile$
  PRINT "opening file ...";outfile$
  OPEN FILE 3,outfile$,WRITE
  base'line:=8100
ENDPROC init
//
PROC write'define'proc
  PRINT FILE 3: "7000 //"
  PRINT FILE 3: "7010 proc define'shap(n)"
  PRINT FILE 3: "7020  for temp=1 to 64"
  PRINT FILE 3: "7030   read temp'data"
  PRINT FILE 3: "7040   image$(temp)=
  chr$(temp'data)"//wrap line
  PRINT FILE 3: "7050  endfor temp"
  PRINT FILE 3: "7060  define n,image$" //2.0
  use ( ) like--> define (n,image$)//wrap line
  PRINT FILE 3: "7070  write'files(chr$(n+65))
  PRINT FILE 3: "7080 endproc define'shap"
ENDPROC write'define'proc
//
PROC write'write'files
  PRINT FILE 3: "8000 //"
  PRINT FILE 3: "8010 proc write'files(letter$)"
  PRINT FILE 3: "8020   write file 3: image$"
  PRINT FILE 3: "8030   open file 2,"0:imag.
  animal-"+letter$,write"//wrap line
  PRINT FILE 3: "8040   write file 2: image$"
  PRINT FILE 3: "8050   close file 2"
  PRINT FILE 3: "8060 endproc write'files"
ENDPROC write'write'files
//
PROC write'sprite'data'lines
  WHILE NOT EOD DO
    READ infile$
    OPEN FILE 2,"imag."+infile$,READ
    READ FILE 2: image$
    PRINT "writing ...";infile$
    PRINT FILE 3: base'line-2;"//"
    PRINT FILE 3: base'line-1;"//";infile$
    FOR x:=0 TO 63 STEP 8 DO
      PRINT FILE 3: base'line+x;"data";
      ORD(image$(x+1)),//wrap line
      FOR y:=2 TO 8 DO PRINT FILE 3:
      ",",ORD(image$(x+y)),//wrap line
      PRINT FILE 3: ""
    ENDFOR x
```

more»

```
    CLOSE FILE 2
    base'line:+60
  ENDWHILE
ENDPROC write'sprite'data'lines
//
PROC write'main'program
  PRINT FILE 3: "6010 //use sprites//2.0"
  PRINT FILE 3: "6020 dim image$ of 64
  PRINT FILE 3: "6030 open file 3,"0:image.
  26animals",write"//wrap line
  PRINT FILE 3: "6040 for x=0 to 25 do
  define'shap(x)"//wrap line
  PRINT FILE 3: "6050 close"
ENDPROC write'main'program
```

Sprite Image Maker Program

(images defined from DATA lines & into files)

This program will define all 26 images as well as write the disk files for each.

```
6010 // use sprites // 2.0 only
6020 dim image$ of 64
6030 open file 3,"0:imag.26animals",write
6040 for x=0 to 25 do define'shap(x)
6050 close
7000 //
7010 proc define'shap(n)
7020  for temp=1 to 64
7030   read temp'data
7040   image$(temp)=chr$(temp'data)
7050  endfor temp
7060  define n,image$//define(n,image$)//2.0
7070  write'files(chr$(n+65))//letter
7080 endproc define'shap
8000 //
8010 proc write'files(letter$)
8020  write file 3: image$
8030  open file 2,"0:imag.animal-"+letter$,write
8040  write file 2: image$
8050  close file 2
8060 endproc write'files
8099 // ant
8100 data 0,0,32,0,1,16,0,0
8108 data 136,0,0,68,0,0,34,0
8116 data 0,18,0,0,10,56,31,6
8124 data 124,127,134,254,127,207,255,255
8132 data 255,255,255,255,255,255,207,255
8140 data 127,192,120,89,192,72,73,96
8148 data 68,69,16,130,36,136,100,34
8156 data 136,36,36,72,18,21,144,0
8159 // bear
8160 data 0,0,0,0,0,0,0,0
8168 data 0,0,0,0,0,0,0,0
8176 data 0,0,0,0,0,0,127,240
8184 data 28,255,252,63,255,254,127,255
8192 data 255,95,255,255,255,255,255,247
8200 data 255,255,3,255,254,3,249,238
8208 data 3,112,238,3,112,238,15,241
8216 data 252,0,0,0,0,0,0,0
8219 // cat
8220 data 0,0,0,2,32,0,3,48
8228 data 0,7,248,0,13,220,0,31
8236 data 254,0,14,60,1,3,240,1
8244 data 1,224,1,1,240,1,7,254
8252 data 1,15,255,193,15,255,225,31
8260 data 255,241,29,255,241,29,223,249
8268 data 29,223,249,29,223,249,29,223
8276 data 249,29,199,255,115,207,255,0
8279 // dog
8280 data 0,0,0,0,1,128,0,1
8288 data 192,0,3,224,0,7,188,0
8296 data 7,252,0,7,192,0,3,128
8304 data 0,31,192,224,255,224,227,255
8312 data 224,199,255,224,199,255,224,199
8320 data 255,224,207,252,96,207,252,96
8328 data 255,248,96,127,240,96,63,255
8336 data 124,255,255,255,255,255,255,0
8339 // elk
8340 data 0,0,0,0,1,131,0,0
8348 data 129,0,0,195,0,0,118,0
8356 data 0,28,0,0,24,0,0,60
8364 data 0,0,119,255,255,255,255,255
8372 data 248,255,255,240,191,255,224,159
8380 data 131,224,158,1,192,142,0,192
8388 data 14,0,192,12,0,192,12,0
8396 data 192,4,0,64,4,0,64,0
8399 // fox
8400 data 0,0,0,0,0,0,0,0
8408 data 0,112,0,0,240,0,0,224
8416 data 0,0,224,0,24,240,0,56
8424 data 120,0,124,63,255,247,63,255
```

more»

```
8432 data 255,127,255,240,124,31,224,248
8440 data 15,192,208,6,192,152,6,224
8448 data 192,7,0,0,0,0,0,0
8456 data 0,0,0,0,0,0,0,0
8459 // goat
8460 data 0,0,0,0,0,0,0,0
8468 data 0,4,0,0,28,0,0,56
8476 data 0,0,48,0,0,120,0,0
8484 data 220,0,0,254,0,0,63,128
8492 data 0,31,255,254,15,255,255,15
8500 data 255,253,7,255,252,7,255,252
8508 data 7,3,220,7,1,156,6,0
8516 data 28,6,0,24,12,0,56,0
8519 // horse
8520 data 0,0,0,0,0,0,0,0
8528 data 8,0,0,28,0,0,60,0
8536 data 0,118,0,0,255,0,1,255
8544 data 255,255,248,255,255,240,191,255
8552 data 240,191,255,224,191,255,224,188
8560 data 3,192,60,1,192,56,1,192
8568 data 56,1,192,56,1,192,24,0
8576 data 192,12,0,96,0,0,0,0
8579 // iguana
8580 data 0,0,0,0,0,0,0,0
8588 data 0,0,0,0,0,0,0,0
8596 data 0,0,106,128,0,250,160,0
8604 data 191,252,0,255,255,255,15,255
8612 data 255,15,255,192,15,255,192,30
8620 data 253,224,24,0,224,24,0,96
8628 data 112,3,192,0,0,0,0,0
8636 data 0,0,0,0,0,0,0,0
8639 // jackass
8640 data 0,0,0,0,0,12,0,0
8648 data 24,0,0,56,0,0,124,0
8656 data 0,246,0,0,255,0,3,255
8664 data 255,255,248,255,255,240,255,255
8672 data 240,191,255,224,191,255,224,191
8680 data 7,192,62,3,192,60,1,192
8688 data 28,1,192,28,1,192,28,0
8696 data 192,6,0,96,0,0,0,0
8699 // kangaroo
8700 data 0,56,0,0,63,0,0,61
8708 data 224,0,63,240,0,63,0,0
8716 data 124,0,0,127,0,0,255,252
8724 data 1,255,140,1,255,0,3,254
8732 data 0,3,254,240,3,255,248,3
8740 data 255,248,3,255,248,3,255,248
8748 data 3,255,240,1,255,224,0,127
8756 data 192,255,255,192,255,255,254,0
8759 // lion
8760 data 0,0,0,0,0,0,0,0
8768 data 0,112,0,0,240,0,0,128
8776 data 0,120,128,1,252,128,3,247
8784 data 224,3,255,96,7,254,63,255
8792 data 248,63,255,240,63,255,224,63
8800 data 255,224,62,7,96,54,3,96
8808 data 55,3,112,56,3,128,0,0
8816 data 0,0,0,0,0,0,0,0
8819 // monkey
8820 data 192,0,124,224,60,28,224,126
8828 data 28,224,87,28,112,127,56,112
8836 data 110,56,112,60,248,127,255,240
8844 data 63,255,192,0,255,129,0,127
8852 data 3,0,63,6,0,63,134,0
8860 data 127,252,1,255,248,7,255,128
8868 data 15,207,0,31,14,0,252,14
8876 data 0,248,14,0,0,62,0,0
8879 // newt
8880 data 0,0,0,0,0,0,0,0
8888 data 0,0,0,0,0,0,0,0
8896 data 0,0,0,0,0,0,0,0
8904 data 0,0,0,255,255,255,191,255
8912 data 255,255,255,192,7,255,0,14
8920 data 3,0,56,14,0,0,0,0
8928 data 0,0,0,0,0,0,0,0
8936 data 0,0,0,0,0,0,0,0
8939 // octopus
8940 data 0,124,0,1,255,0,7,255
8948 data 192,7,255,224,15,255,224,15
8956 data 255,224,15,255,240,15,255,240
8964 data 15,255,247,30,49,255,126,181
8972 data 252,255,255,240,199,239,255,143
8980 data 239,227,25,255,225,51,179,113
8988 data 115,51,49,99,49,176,99,49
8996 data 152,99,49,143,111,30,243,0
8999 // pig
9000 data 0,0,0,0,0,0,0,0
9008 data 0,0,0,0,12,0,0,30
9016 data 63,240,63,255,252,239,255,254
9024 data 255,255,255,255,255,254,127,255
9032 data 254,7,255,254,7,255,254,3
9040 data 127,246,3,96,54,7,96,54
```

more»

```
9048 data 0,192,108,0,0,0,0,0
9056 data 0,0,0,0,0,0,0,0
9059 // quail
9060 data 0,0,0,0,0,0,0,0
9068 data 0,48,0,0,48,0,0,120
9076 data 0,0,220,0,0,248,0,0
9084 data 60,0,0,31,128,14,31,224
9092 data 28,15,255,248,15,255,240,7
9100 data 255,128,3,254,0,0,252,0
9108 data 0,80,0,0,80,0,0,160
9116 data 0,0,0,0,0,0,0,0
9119 // rooster
9120 data 0,0,56,0,0,124,96,0
9128 data 124,248,0,236,252,1,255,255
9136 data 255,252,255,255,236,255,255,204
9144 data 207,255,192,231,255,128,225,255
9152 data 0,225,254,0,96,252,0,0
9160 data 108,0,0,108,0,0,108,0
9168 data 0,255,0,0,0,0,0,0
9176 data 0,0,0,0,0,0,0,0
9179 // squirrel
9180 data 12,0,0,28,0,0,60,0
9188 data 7,94,0,31,255,128,63,255
9196 data 224,127,31,248,127,15,254,127
9204 data 15,255,127,255,255,63,3,255
9212 data 159,3,255,223,3,255,223,7
9220 data 255,223,7,255,223,7,255,191
9228 data 7,255,255,3,255,254,3,255
9236 data 252,1,255,192,15,255,192,0
9239 // turkey
9240 data 0,0,0,0,0,0,0,0
9248 data 24,0,0,60,0,0,118,192
9256 data 0,127,225,248,124,243,254,244
9264 data 255,255,228,255,255,224,127,255
9272 data 224,7,255,224,3,255,192,1
9280 data 255,0,0,108,0,0,108,0
9288 data 0,108,0,0,103,0,0,56
9296 data 0,0,0,0,0,0,0,0
9299 // urchin
9300 data 0,0,0,0,0,0,0,0
9308 data 0,0,0,0,0,0,0,0
9316 data 0,0,0,0,0,1,74,128
9324 data 11,255,160,47,255,232,63,255
9332 data 250,191,255,254,255,255,255,255
9340 data 255,255,0,0,0,0,0,0
9348 data 0,0,0,0,0,0,0,0
9356 data 0,0,0,0,0,0,0,0
9359 // vulture
9360 data 0,0,0,252,0,0,238,0
9368 data 0,62,0,0,60,0,0,28
9376 data 0,0,14,248,0,7,252,0
9384 data 7,255,0,7,255,128,15,255
9392 data 128,15,255,192,15,255,192,15
9400 data 255,224,15,255,224,7,255,224
9408 data 7,255,224,3,79,192,3,67
9416 data 192,3,67,192,7,193,192,0
9419 // whale
9420 data 0,0,0,0,0,0,0,0
9428 data 0,0,0,0,0,0,0,0
9436 data 3,248,128,31,252,192,127,254
9444 data 225,255,255,127,255,255,255,255
9452 data 239,224,255,254,192,31,252,0
9460 data 0,0,0,0,0,0,0,0
9468 data 0,0,0,0,0,0,0,0
9476 data 0,0,0,0,0,0,0,0
9479 // xenopus
9480 data 216,56,48,248,124,182,112,186
9488 data 252,96,254,56,112,254,28,112
9496 data 124,28,63,255,220,61,255,252
9504 data 3,255,184,131,255,193,179,255
9512 data 205,227,255,199,195,255,195,221
9520 data 255,187,223,255,251,223,255,251
9528 data 249,255,159,240,254,14,0,56
9536 data 0,0,0,0,0,0,0,0
9539 // yak
9540 data 0,0,0,120,0,0,96,0
9548 data 0,96,0,0,125,240,0,63
9556 data 252,0,95,255,240,255,255,254
9564 data 255,255,255,7,255,255,7,255
9572 data 255,3,255,255,1,255,255,0
9580 data 253,254,0,240,188,0,224,60
9588 data 0,192,48,1,192,112,0,0
9596 data 0,0,0,0,0,0,0,0
9599 // zebra
9600 data 0,0,0,0,0,0,0,0
9608 data 8,0,0,28,0,0,60,0
9616 data 0,118,0,0,223,0,1,87
9624 data 239,106,120,210,74,208,182,210
9632 data 176,173,182,160,171,117,96,180
9640 data 3,64,44,1,192,24,1,64
9648 data 48,1,64,40,1,192,24,0
9656 data 192,12,0,96,0,0,0,0 ■
```

Sprite Editor

by David Warman

This is an upgraded version of the sprite editor found on Cartridge Demo disk #3. It has **many** added features, including the ability to animate a sprite. Here is a list of the keys used in editing a sprite image:

CURSOR KEYS - Used to move around the edit screen.

SPACE BAR - Toggles a pixel on and off. If you are in multicolor mode, the space bar toggles the pixel between the background color (off) and color 1. To get color 2 or color 3, press «2» or «3» respectively.

«CTRL»+0 - Selects background color. Choose 0-15 for the color you want.

«CTRL»+1 - Selects sprite color 1. Choose 0-15 for the color you want.

«CTRL»+2 - Selects sprite color 2. Choose 0-15 for the color you want.

«CTRL»+3 - Selects sprite color 3. Choose 0-15 for the color you want.

^ - Toggles the column the cursor is on. All pixels that are on are turned off, and all pixels that are off are turned on.

<- - (left arrow) Toggles row the cursor is on.

n - Makes a negative of the entire sprite, All pixels that are on are turned off and vice versa.

h - Flips the sprite in the horizontal direction, i.e. the sprite is spun on its vertical axis.

v - Flips the sprite vertically (turns it upside-down).

f1 - Moves the sprite up one pixel. The top row is wrapped around to the bottom.

f7 - Moves the sprite image down, with wrap in effect.

f3 - Moves the sprite image left, with wrap.

f5 - Moves the sprite image right, with wrap.

«HOME» - Puts the cursor in the home position.

«CLR» - Erases the sprite.

m - Toggles between multicolor and hi-res sprite display.

e - Lets you select a sprite image to edit. This program allows you to work with up to eight different sprite images at once. The eight images are shown in the windows, or cells, at the top of the screen numbered 0-7 (only four are displayed at once). Pressing **e** lets you select which image to edit; the default is image zero when the program starts. If the cell you select contains an image, it is copied into the large editing area, otherwise the editing area is cleared. In either case, the image you were working on is stored in one of the empty cells, or back into the cell it came out of.

***** - Toggles between displaying images 0-3 and 4-7 in the cells at the top of the screen.

a - Animates a group of images. After pressing **a**, you can choose which images you want to animate by typing the cell numbers in the order you want them shown. Hit «return» after choosing the last image. You can then go on editing and watch the animation image change as you change the shape (assuming the image you are currently editing is one of the images used in the animation sequence). The animation can be stopped by pressing **a** again.

+ - Increases animation speed.

- - Decreases animation speed.

d - Deletes the image in the cell you choose.

c - Copies an image from one cell to another.

s - Saves the cell image you select to disk.

l - Loads an image from disk into an empty cell. If the filename of the sprite begins or ends with **shap**, it can be loaded by moving a pointer to it and pressing «return». If the filename does not contain **shap**, use the command below.

L - Loads an image from disk into an empty cell. You must type in the filename.

$ - Reads the directory of a new disk. This command allows shape files beginning or ending with **shap** to be loaded as mentioned above. It is not useful on disks without **shap** files.

q - Exits program.

i - Displays a summary of these instructions. ■

Buffer Editor

by David Warman

I have found the buffer package to be very useful (this package is in the book disk set *Packages Library Vol 1*). I've used it in several programs to store large amounts of text or data that wouldn't fit in the main memory. The only trouble with using the buffer is getting the information in there in the first place, or worse, trying to edit it later. This program provides an easy way to edit the buffer and allows you to see exactly what is in each location in the buffer. The first time the program is run, it attempts to load the two files: "*txt.buffer inst*" and "*txt.buffer help*" into the buffer. The first file contains complete instructions on how to use the program (much of this text is also reprinted below). The second file contains prompts that are displayed in various parts of the program and is a good example of how prompts can be stored in the buffer and retrieved by a program. The program will run without these files, but there would then be no instructions or prompts.

The buffer package is linked onto this program on *Today Disk #22*. Without the buffer package this program will not run. Thus, this program cannot be a type in program and is not listed in this issue. The program, however, is excellent, and may give you an added reason to get the matching *Today Disk #22*.

By the way, the help file for the Directory Designer (also in this issue) was created with this buffer editor.

Here is a listing of most of the instructions that are included on the disk with the program:

Use the cursor up/down keys to read these instructions. Hit the «home» key from the home position to return to the start of the buffer.

The buffer package is very useful for storing large amounts of data or text. Help screens can be put in the buffer without using up any program memory. (This can be done with the text package too, but the buffer package allows instant access to any point in the buffer - like a random access file.) But putting the information in the buffer in the first place can sometimes be a hassle, especially if you have to edit it later.

This program makes it easier by allowing you to visually place text anywhere in the buffer. Simply type the text on the screen exactly as you want it to appear in the buffer. Don't worry about breaking words on the end of the line; a PROCedure to read the buffer will display the text properly. Your program can read a specific area of text from the buffer by referring to the starting and ending buffer locations.

For example: This paragraph starts at location 5070 and ends at 5805. (To prove it, move the cursor to the ends of this paragraph and note the buffer position, shown below this window.)

The command **rdbuffer(a$,start,end-start+1)** *will read this paragraph and store it in the variable* **a$**, *provided that* **a$** *is DIMensioned sufficiently high. In this case* **start** *is 5070 and* **end** *is 5805. After reading part of the buffer into* **a$**, *a PROCedure (such as* **print'buffer** *in this program) can be used to display the text in an area on the screen, with word-wrap in effect. To see how this works, press «f5» to extend the buffer past 9400 bytes. Then scroll down to buffer location 9500. This is where the prompts for this program are stored. They are read by the PROCedure* **print'buffer** *and displayed in a window at the top of the screen.*

To see it in action, try pressing «CLR». A message from the buffer will be displayed asking if you really want to clear the window or buffer. Don't clear it unless you want to erase part or all of the buffer!

more»

Command Descriptions

F1 - *allows you to enter the chr$ code of unprintable characters. Things like return, color codes, insert, delete, etc. can be put in the buffer but will not show up on the screen. This program shows unprintable character codes as -. The ORD value of these characters, as well as all others, is shown above this window.*

F7 - *determines how unprintable codes will be printed. When on, unprintable codes are shown as -. When off, they are printed as themselves, so if something like a delete, a color code, or a clear screen is encountered in the buffer, it will actually be printed. Obviously this can severely mess up the screen display, but the advantage is that printing is much faster with the translator off. If you make typing mistakes and must use insert or delete, it is advisable to turn the translator off first; this will speed things up considerably. If the screen does get messed up by printing certain characters (like «home»), it can be fixed by stopping the program and typing* **RUN.** *This will not disturb the contents of the buffer.*

F3 - *displays these instructions.*

F5 - *jumps to a certain location in the buffer. You are asked what buffer position you want to jump to. The edit window then shows the part of the buffer starting at the position you chose, or the screen line starting nearest that position. (The buffer is shown in this window so that position 1 is always on the left side of the screen.)*

F2 - *toggles the maximum buffer size between 9400 and 10,000 bytes. When it is 9400 bytes, the prompt messages that appear at the top of the screen are protected from being overwritten or moved by insert or delete. If you need to use all 10,000 bytes, you can press* **F2** *and overwrite the prompts, but then they will not be available for use in this program. (Actually, the maximum buffer size is about 16,000 bytes, but the windows package which is used in this program uses up some of the memory. Any other package that links in under the cartridge will restrict the buffer space, or may not even work with the buffer. Packages that don't load under the cartridge - like rabbit - will not reduce buffer room.)*

RUN/STOP - *allows you to save an area of the buffer, load a seq file into the buffer, or quit.*

F6/F8 - *shrinks/widens window size. If you use the print'buffer PROCedure in your program to display the buffer contents, you can preview just how it will look by setting this window to the same width as the one in your program. NOTE: You can erase these instructions after you have read them by pressing «CLR» and* **e**. *This will not disturb the prompts, as long as the number next to* **max** *is 9400.*

Further Reference:

Edit Source, Dick Klingens, #18 pg 59
Help Screen Editor, Dick Klingens, #17 pg 72
Screen Editor Revisited, Gerald Hobart, #17 pg 74
Text Input Window, David Stidolph, #16 pg 16
Screen Help Package, Gerald Hobart, #16 pg 18
CAI Editor, Reed Brown, #16 pg 22
Easy Instructions, Captain COMAL, #16 pg 24
Smart File Reader, Captain COMAL, #16 pg 29
Instant Help Screens, Dick Klingens, #15 pg 61
Dual Screen, Craig Hardy, #14 pg 25
Text Package, Dick Klingens, #11 pg 63
2.0 Scrolling Text, Dick Klingens, #8 pg 59
Packages Library Vol 1, Buffer Package, pg 6
Packages Library Vol 2, Windows Package, pg 53 ■

Maze

by Bill Inhelder

In a recent issue of the Smithsonian magazine there is an article describing Japan's maze craze. There are over 100 commercial labyrinths in Japanese amusement complexes where people eagerly pay money and spend an average of 50 minutes attempting to run such mazes.

The article called to mind an old program, which randomly generates a fairly complex maze and then simulates the run of a mouse whose objective is to find the cheese hidden in the maze. To run most mazes successsfully, the fundamental rule is to hold your left hand (or right hand) on the wall as you walk and you will always find the exit. It may not be the shortest path to the exit but you will get there. The rule works well even with mazes which have internal free standing walls.

Using this left hand rule, the program *mouse maze* (listed after this article and on *Today Disk #22*) will produce mouse runs of 40 seconds or less depending on the complexity of the maze. Watching the path of the mouse from above and the many wrong turns the mouse takes, it is hard for us to imagine how difficult and frustrating it would be if we were actually running the maze at ground level without using the left hand rule. The Japanese remind their customers to go to the bathroom before entering the maze! If panic sets in during the run, red flags are available which are waved over the head to call for assistance to be led out of the maze.

In *mouse'maze'alt* (on *Today Disk #22*), the mouse uses a randomly selected path at each intersection and will only reverse direction at the end of a blind alley. The mouse has no memory and will run some of the same paths over and over again before finding the cheese. In 10 trial runs the times ranged from 12 to 424 seconds with a median value of 60 seconds. Luck and maze complexity explain the wide range of times.

Writing the program in COMAL was both challenging and fun. The challenging part was trying to unscramble [and understand] the many BASIC GOTO's in the old program and establish some structure using procedure calls, loops, whiles, and recursive calls of a procedure. It runs under Power Driver and 2.0.

Mouse Maze

```
instructions
DIM dr(0:4), ans$ OF 1
REPEAT
  initialize
  rev'screen
  const'maze'a
  POKE 53272,21
  start'mouse
  WHILE KEY$="" DO NULL
  PRINT AT 1,1: "              000000"
  TIME 0
  PRINT AT 1,1:"elapsed time:",str$(time div 60)
  run'maze
  WHILE KEY$="" DO NULL
  PAGE
  POKE 53272,23
  PRINT AT 12,1: "Enter s to stop; or c to"
  INPUT "repeat with another maze: ": ans$
UNTIL ans$ IN "Ss"
POKE 53272,23
PAGE
//
PROC initialize
  dr(0):=2; dr(1):=-80; dr(2):=-2; dr(3):=80
  mz'wall:=160; mz'path:=32; scr:=1024
  pos:=scr+82; flag:=0
  flag2:=0; flag3:=0
  PAGE
ENDPROC initialize
//
PROC rev'screen
  PRINT
  FOR i:=1 TO 23 DO
```

more»

```
    PRINT " ",CHR$(18),
    FOR g:=1 TO 37 DO PRINT CHR$(32),
    PRINT CHR$(146)
  ENDFOR i
  POKE pos,4
ENDPROC rev'screen
//
PROC const'maze'a
  j:=RND(0,3)
  x:=j
  REPEAT
    b:=pos+dr(j)
    IF PEEK(b)=mz'wall THEN
      POKE b,j
      POKE pos+dr(j)/2,mz'path
      pos:=b
      const'maze'a
    ELSE
      j:=(j+1)*(j<3)
    ENDIF
  UNTIL j=x
  IF flag=0 THEN
    const'maze'b
  ELSE
    IF flag2=0 THEN const'maze'c
  ENDIF
ENDPROC const'maze'a
//
PROC const'maze'b
  IF (pos>1844 AND pos<1864) OR (pos>1924
  AND pos<1943) THEN//wrap line
    c:=pos
    flag:=1
  ENDIF
  const'maze'c
ENDPROC const'maze'b
//
PROC const'maze'c
  j:=PEEK(pos)
  POKE pos,mz'path
  IF j<4 THEN
    pos:=pos-dr(j)
    const'maze'a
  ENDIF
  POKE c,42
  POKE 54272+c,3
  flag2:=1
ENDPROC const'maze'c
//
PROC run'maze
  WHILE flag3=0 DO
    PRINT AT 1,19: STR$(TIME DIV 60)
    b:=pos+dr(j)/2
    IF PEEK(b)=42 THEN
      flag3:=1
    ELSE
      IF PEEK(b)=mz'path THEN
        POKE b,81
        POKE 54272+b,15
        POKE pos,mz'path
        pos:=b
        j:=(j+2)-4*(j>1)
        j:=(j-1)+4*(j=0)
      ELSE
        j:=(j-1)+4*(j=0)
      ENDIF
    ENDIF
  ENDWHILE
ENDPROC run'maze
//
PROC start'mouse
  POKE pos,81
  POKE 54272+pos,1
  j:=2
ENDPROC start'mouse
//
PROC instructions
  PAGE
  PRINT AT 2,15: "MOUSE MAZE"
  PRINT AT 5,20: "By Bill Inhelder"
  PRINT AT 9,1: "After the random maze is"
  PRINT "completed and the cheese (*) and the"
  PRINT "mouse (a white circle) are placed in"
  PRINT "the maze, press any key to activate"
  PRINT "the mouse. When the mouse finds the"
  PRINT "cheese, press any key again and"
  PRINT "follow screen instructions."
  PRINT AT 23,5: "Press    C    to begin"
  WHILE KEY$<>"c" DO NULL
ENDPROC instructions ■
```

Directory Designer

by David Warman

This COMAL 2.0 program helps you arrange a disk directory the way you want it. Add comments to the directory, trace a file, recover a scratched file, even edit the BAM. Since the program uses special packages and buffer text it cannot be a type in program listing, however it is on *Today Disk #22*. Here is a complete list of the commands that are available.

Command Summary

CURSOR KEYS - Use these keys to scroll through the directory.

«home» - Moves to the first directory entry.

«clr» - Moves to the last directory entry.

«return» - When you press the *«return»* key, the filename that the pointer is pointing to will be picked up and placed in a temporary buffer. You can then scroll through the directory and drop that filename wherever you want by pressing «return» again.

s - Starts the sort option. After pressing **s**, you can scroll through the directory, hilighting the range you wish to sort. Then press **s** again to perform the sort. Or, instead of hilighting a range to sort, press **f** to sort from the pointer to the first entry, **l** to sort from the pointer to the last entry, **a** to sort all entries, or **<-** (left arrow) to abort the sort.

S - Pressing **shift-s** toggles between ascending and descending sort. The program defaults to sorting in ascending order.

e - Hit **e** to edit a filename. Change the filename, then hit *«return»*. A small HELP window also pops up reminding you that the function keys are pre-set, and displays the settings in the window. Instead of editing the filename you may press one of the function keys, predefined as a box top, sides, middle, and bottom making it easy to insert a title box into the directory. Since the function keys are defined in the program you can change them as you see fit. They are defined in the **INIT** procedure. However, the HELP window won't reflect your changes, since it comes from the buffer area. Just ignore the window prompt or use the buffer editor (on *Today Disk #22*) to correct the prompts.

i or **«space bar»** - Either of these keys will insert a zero-block USR file with the filename "■■■■■■■■■■■■■■■■" into the directory. This filename can be left as is or edited to put comments in the directory, like Today disks.

r or **«del»** - Either of these keys will remove a filename permanently from the directory.

c - Makes a copy of a filename. The copy can be placed anywhere in the directory.

d - Deletes an entry from the directory; the same as the COMAL **DELETE** command. The filename will still be in the directory, but the status (filetype) is changed to DEL and it will not show up with a CAT command.

u - Opposite of **d**. A file that has been deleted can be unscratched with this command. If nothing has been saved to the disk since the file was deleted, it should be recoverable by unscratching it and then either validating the disk, or loading the program and re-saving it. Even if something has been saved to the disk since deleting a program, it can sometimes be recovered by this method.

p - Protects a file so that it cannot be deleted. The filename will be shown with a "<" next to the file type in the directory listing.

more»

t - This changes the file type. Repeatedly pushing t will cycle the file type through SEQ, PRG, USR, and REL.

w - Changes the size of the window that the directory is displayed in. The normal window just shows the filename and type, like CAT does. The wide window also shows the starting track and sector, and the number of blocks that the file uses, but the scrolling is slower.

@ - Find and replace. You type in a string to search for and a string to replace it with. All occurences of the first string are replaced by the second string.

h - Displays a list of these commands.

Special STOP key commands

This program uses the popover menu system from *COMAL Today #11*. Pressing the «STOP» key displays a menu with these options:

r - Reads the directory of the disk in drive 8 into memory. This will undo any changes that have not been written to the disk.

w - Writes the files in memory to the disk in drive 8.

c - This option lets you change the disk name.

t - Traces the file pointed to by the pointer. Each sector used by the file is printed.

b - Displays and allows you to edit the BAM (Block Availability Map). The blocks on the disk that are used are represented by solid circles, unused blocks are indicated by periods. The SPACE BAR toggles the block between allocated and de-allocated. When finished with the BAM, press *«return»* to return to the normal screen display.

q - Quits program.

The program also comes with a nice HELP system that reminds you of commands and provides some advice. For instance, when trying to add a new directory entry when there is no room, this message will pop up:

Unable to find room for added directory entries. Try checking BAM and de-allocating blocks on track 18.

Or, if you choose to read in a new directory after you have modified a current one without saving it back, you will be reminded with this message in a pop up window:

Reading the directory will cancel any changes you have made. Are you sure?

Further Reference:

Reorder Directory, Phyrne Bacon, this issue
Unerase Files, Bob McCauley, #20 pg 11
File Recovery, Christopher Laprise, #19 pg 15
Directory Boxes, Paul Keck, #19 pg 68
Dummy Files, Jack Baldrige, #16 pg 27
Read Directory, David Stidolph, #16 pg 57
Disk Editor, Phyrne Bacon, #16 pg 59
Directory Probe, Phyrne Bacon, #16 pg 59
Directory Sort, Jack Baldrige, #14 pg 36
Disk Editor, Phyrne Bacon, #13 pg 28
Unscratching Files, Phyrne Bacon, #13 pg 31
Fast Dir Revisited, Ray Carter, #12 pg 50
Fix Disk Errors, David Stidolph, #11 pg 16
Your RUN/POP key, Len Lindsay, #11 pg 18
2.0 Disk Directory Tips, #11 pg 22
DIR for COMAL 0.14, David Stidolph, #10 pg 28
Function Keys, Mike Lawrence, #9 pg 33
Directory Editor, Robert Ross, #8 pg 55
Fast Directory, #8 pg 68
Redefine Function Keys, #7 pg 21
Disk Editor, Ian MacPhedran, #7 pg 56;#5 pg 45
Disk Protector, Glen Colbert, #4 pg 17
Disk Directory Manipulator, David Stidolph, #3 pg 6 ■

Reorder Directory

by Phyrne Youens Bacon

The program *reorder'dir* can be used to edit a disk directory. It can do any of the following:

- change the order in which the files are listed
- add empty directory entries
- delete directory entries
- create dummy files
- change filenames
- change filetypes listed

Back up the disk to be edited, and work only on the backup. That way, you can start over if you run into difficulty.

When the program is run, the whole disk directory is loaded into memory and listed on the screen. A directory entry might be listed as follows:

35 prg reorder'dir 17-1 18-1

35 is the number of blocks in the file
prg is the filetype
reorder'dir is the filename
17-1 is the first block of the file
18-1 is the block the entry would be in if the current version of the directory were saved to disk by reorder'dir.

Since entries for some filetypes, such as relative files, contain additional information, the whole thirty-two byte entry is stored in memory and will be resaved to disk in its chosen position in the directory when f8 is pressed. However, the program can change some information in an entry, such as the filename (see below).

Edit Commands

Press f to change filename, or to put a filename on a blank entry.
Press y to change file type, and then press:
  s for seq
  S for protected seq
  p for prg
  P for protected prg
  d for deleted (scratched) file
  D for protected deleted (scratched) file
  u for usr
  U for protected usr

Press b to change the filelength listed in the directory, and enter a decimal number. This does not change the actual number of blocks in the file.
Press t to change the first block track number, and enter a decimal number
Press s to change the sector number. Since these do not change the track/sector of the first block of the file, the track/sector change should only be used to create dummy files (see below).
Press p to send the version of the directory on the screen to the printer. If you need to see every byte of some directory, use a disk editor.
Press m to save a directory entry in temporary memory: this deletes it from the directory for the time being. Temporary memory can hold the full directory: up to 144 entries.
Press r to recall all temporarily saved entries to above the present location. The recalled entries will appear in the order they were loaded into temporary memory. Recalling a small number of entries from temporary memory is fast, but it may take a few minutes to recall a very large number of entries. Be sure to press r (recall) before saving the directory.
Press f1 to insert a blank entry above present location.
Press f2 to delete the entry at present location.
Press f3 for a help screen with all these commands listed.
Press f4 to exit.
Press f5 to move top of the current screen display up 12 filenames.
Press f6 to display last filename.
Press «home» to go to the top of the screen.

more»

Press «home» twice to go to first filename.
Press f7 to move top of the current screen display down 12 filenames.
Press f8 to save the revised directory.

To reorder all or part of the directory, press m beside each entry in the order you want them to appear, and then when you have them loaded into temporary memory in the right order, press r when at the right location to restore the entries to the directory.

If you erase an entry in error, just press the «stop» key, and rerun the program from the start. Nothing is saved to disk until f8 is pressed, and the question "**save directory: are you sure?**" is answered **yes.**

When the directory is saved, **track 18 sector 18** is also saved with the hex link address **00 ff.** With this link address, **18-18** will be accepted by pass"v" as the first (and last) block of a legitimate file, and **18-18** can be used as the first track/sector of any dummy file entry; for example, of the entry:

```
 0  "++++++++++++++++" usr
```

To create a dummy file:

Press f1 to get a blank entry
Press f to enter the filename
Press y to enter the filetype
  (press u for usr)
Press b to enter the filelength
  (enter 0 for zero blocks)
Press t and enter 18
Press s and enter 18

Whenever a directory is saved by a Commodore disk drive or by this program, the unused entries in the last block of the directory contain thirty-two zeros: chr$(0)'s.

If you add entries to the directory, you may wish to remove unused entries and scratched entries to reduce the number of blocks in the directory. The block that each entry will be saved in is listed in the right hand column; so it is easy to tell whether the number of blocks in the directory can be reduced.

It is always a good idea to use pass"v" after using *reorder'dir*, and to check afterwards to see that all the files are still listed, and are in the proper order. For a list of further references, see *Directory Designer* in this issue.

```
// delete "reorderdir2"
//   save "reorderdir2"
//reorder'dir by phyrne bacon march, 1988
DIM e$(1:160) OF 33, z$ OF 1, d$ OF 1
DIM b$ OF 256, bb$ OF 256, no$ OF 33
DIM mm$(1:144) OF 33, fn$ OF 16, bk$ OF 33
intro
init
init2
read'in'dir
act
END
FUNC dir'sec(i)
  IF i<7 THEN
    j:=1+3*(i-1)
  ELIF i<13 THEN
    j:=2+3*(i-7)
  ELIF i<19 THEN
    j:=3+3*(i-13)
  ELSE
    j:=99
  ENDIF
  RETURN j
ENDFUNC dir'sec
PROC init
  bk$:=""; z$:=CHR$(0); d$:=CHR$(18); bb$:=""
  mem:=0; pp:=0
  FOR i:=1 TO 33 DO bk$:=bk$+z$
  no$:=bk$
  no$(6:21):="----------------"
  nu$:=no$
  nu$(3:5):=CHR$($83)+CHR$(18)+CHR$(18)
  FOR i:=1 TO 8 DO bb$:=bb$+bk$(1:32)
  FOR i:=1 TO 160 DO
```

more»

```
    e$(i):=no$
  ENDFOR i
  ne:=0
ENDPROC init
PROC output'dir
  j:=ne DIV 8
  IF ne MOD 8<>0 THEN j:+1
  IF j=0 THEN j:=1
  FOR n:=1 TO j DO output'block(n,j)
  IF sec<>18 THEN
    b$:=""
    FOR i:=1 TO 8 DO b$:=b$+bk$
    b$(2):=CHR$(255)
    wrbk(18,18)
  ENDIF
ENDPROC output'dir
PROC output'block(n,j)
  b$:=""
  FOR i:=1 TO 8 DO
    IF n*8-8+i<=ne THEN
      b$:=b$+e$(n*8-8+i)(1:32)
    ELSE
      b$:=b$+bk$(1:32)
    ENDIF
  ENDFOR i
  IF n=j THEN
    b$(1):=z$; b$(2):=CHR$(255)
  ELSE
    b$(1):=d$; b$(2):=CHR$(dir'sec(n+1))
  ENDIF
  tr:=18; sec:=dir'sec(n)
  wrbk(tr,sec)
ENDPROC output'block
PROC blck(REF tr,REF sec)
  TRAP
    PASS "i0"
    OPEN FILE 7,"u8:#/s2",READ
    PASS "u1:2 0 18 "+STR$(sec)+" "
    b$:=GET$(7,256)
    CLOSE FILE 7
  HANDLER
    CLOSE FILE 7
    PAGE
    PRINT AT 23,1: ERRTEXT$
    END
  ENDTRAP
  nt:=ORD(b$(1)); ns:=ORD(b$(2))
ENDPROC blck
PROC wrbk(tr,sec)
  TRAP
    OPEN FILE 7,"u8:#/s2",WRITE
    PASS "b-p: 2 0 "
    PRINT FILE 7: b$,
    PASS "u2: 2 0 18 "+STR$(sec)+" "
    CLOSE FILE 7
    IF tr=18 THEN PASS "i0"
  HANDLER
    CLOSE FILE 7
    cltp
    PRINT AT 23,1: ERRTEXT$
    getk
  ENDTRAP
ENDPROC wrbk
PROC getk
  REPEAT k$:=KEY$ UNTIL k$<>""
ENDPROC getk
PROC read'in'dir
  b$:=bb$; j:=0; ne:=0; nt:=18; ns:=0
  blck(nt,ns)
  dkname$:=b$(145:160); dkid$:=b$(163:164)
  WHILE nt<>0 DO
    blck(nt,ns)
    b$(1:2):=z$+z$
    FOR i:=0 TO 7 DO
      e$(j*8+i+1):=b$(i*32+1:(i+1)*32)
      ee(j*8+i+1)
    ENDFOR i
    ne:+8; j:+1
  ENDWHILE
ENDPROC read'in'dir
PROC init2
  DIM qu$ OF 1, n$ OF 1, f$ OF 1
  DIM dkname$ OF 16, dkid$ OF 2
  DIM l$(1:145) OF 39, a$ OF 39, cr$ OF 1
  DIM tl$ OF 1, x$ OF 16
  DIM fname$ OF 16, ds$ OF 40, t$ OF 1
  DIM zt$ OF 1, pname$(1:22) OF 16
  PRINT ""147""17" loading directory..."
  r:=1; c:=1; zt$:=CHR$(0); cr$:=CHR$(13)
  tl$:=CHR$(128)
  qu$:=CHR$(34); sn:=0
  n$:=CHR$(18); f$:=CHR$(146)
```

more»

```
ENDPROC init2
PROC act
  k$:=""; sn:=0; on:=-1
  nb
  REPEAT
    IF sn<>on THEN pag
    h:=0
    REPEAT t$:=KEY$ UNTIL t$=zt$
    REPEAT
      IF h=0 THEN PRINT AT r,10: ">"
      IF h=14 THEN PRINT AT r,10: " "
      h:+1; t$:=KEY$
      IF h=28 THEN h:=0
    UNTIL t$<>zt$
    IF h<15 THEN PRINT AT r,10: " "
    cn:=0
    IF t$=""17"" AND sn+r<=ne THEN
      r:+1; cn:=1
      pinc
    ELIF t$=""145"" AND sn+r<>1 THEN
      r:-1; cn:=1
      IF r=0 THEN
        r:=1; sn:-1
        IF sn<0 THEN sn:=0
        pag
      ENDIF
    ELIF t$=cr$ AND sn+r<=ne+1 THEN
      r:+1; c:=1; cn:=1
      pinc
    ELIF t$=""19"" AND sn+r<>1 THEN
      IF r=1 AND c=1 THEN
        sn:=0
        nb
        pag
      ENDIF
      r:=1; c:=1; cn:=1
    ELIF t$=qu$ THEN
    ELIF t$=""135"" AND sn>0 THEN
      pp:=sn+r; sn:-12
      pag
    ELIF t$=""136"" AND sn<ne-20 THEN
      pp:=sn+r; sn:+12
      pag
    ELIF t$=""139"" AND sn<ne-20 THEN
      pp:=sn+r; sn:=ne-20
      pag
      nb
    ELIF t$=""140"" AND sn+r<=ne THEN
      cltp
      INPUT AT 23,1: "save directory: are you
      sure? y"157"": k$ //wrap line
      IF k$="y" THEN
        output'dir
      ENDIF
      nb
    ELIF t$=""133"" AND sn+r<=ne+1 THEN
      j:=sn+r; i:=ne
      IF ne<144 THEN
        WHILE i>=j DO
          e$(i+1):=e$(i); i:-1
        ENDWHILE
        e$(j):=bk$; ne:+1
        pag
      ELSE
        PRINT AT 23,1: "too many entries in
        directory.        "//wrap line
        PRINT AT 24,1: "use f2 to delete an
        entry             "//wrap line
        PRINT AT 25,1: "before trying to inset
        a new entry.    ",//wrap line
        getk
        nb
      ENDIF
    ELIF t$=""137"" AND sn+r<=ne THEN
      j:=sn+r; i:=j
      WHILE i<ne DO e$(i):=e$(i+1); i:+1
      e$(ne):=no$; ne:-1
      pag
    ELIF t$="m" AND sn+r<=ne THEN
      mem:+1; mm$(mem):=e$(sn+r)
      j:=sn+r; i:=j
      WHILE i<ne DO e$(i):=e$(i+1); i:+1
      e$(ne):=no$; ne:-1
      pag
    ELIF t$="r" and sn+r<=ne+1 and mem>0 then
      IF ne+mem<=144 THEN
        i:=ne; ne:+mem
        WHILE i>=sn+r DO
          e$(i+mem):=e$(i); i:-1
        ENDWHILE
        FOR i:=1 TO mem DO
          e$(sn+r+i-1):=mm$(i)
```

more»

```
    ENDFOR i
    mem:=0
    pag
  ELSE
    PRINT AT 23,1: "too many entries in
    directory.          "//wrap line
    PRINT AT 24,1: "use f2 to delete
    ",ne+mem-144," entries"//wrap line
    PRINT AT 25,1: "and then press r again
                       ",//wrap line
    getk
    nb
  ENDIF
ELIF t$=""134"" THEN
  help
  REPEAT
    k$:=KEY$
  UNTIL k$<>CHR$(0)
  PRINT ""147"",
  nb
  pag
ELIF t$=""147"" THEN
  pag
ELIF t$=""138"" THEN
  cltp
  INPUT AT 23,1: "exit: are you
  sure? y"157"": k$//wrap line
  IF k$="y" THEN
    END
  ENDIF
  nb
ELIF t$="y" AND sn+r<=ne THEN
  PRINT AT r,10: ">"
  cltp
  INPUT AT 23,1: "filetype?
  (p,s,u,d,r,P,S,U,D,R) p"157"": k$//wrap line
  CASE k$ OF
  WHEN "d"
    i:=$00
  WHEN "D"
    i:=$80
  WHEN "p","P"
    i:=$82
  WHEN "s","S"
    i:=$81
  WHEN "u","U"
    i:=$83
  WHEN "r","R"
    i:=$84
  OTHERWISE
    i:=$81
  ENDCASE
  IF ORD(k$)>96 THEN i:=i+$40
  e$(sn+r)(3):=CHR$(i)
  reprt(r)
  nb
ELIF t$="f" AND sn+r<=ne THEN
  PRINT AT r,10: ">"
  cltp
  INPUT AT 23,1: "filename? ": fn$
  WHILE len(fn$)<16 DO fn$:=fn$+CHR$(160)
  e$(sn+r)(6:21):=fn$
  ee(sn+r)
  reprt(r)
  nb
ELIF t$="b" AND sn+r<=ne THEN
  cltp
  PRINT AT r,10: ">"
  INPUT AT 23,1: "filelength? ": i
  e$(sn+r)(31):=CHR$(i MOD 256)
  e$(sn+r)(32):=CHR$(i DIV 256)
  reprt(r)
  nb
ELIF t$="t" AND sn+r<=ne THEN
  cltp
  PRINT AT r,10: ">"
  INPUT AT 23,1: "track? ": i
  e$(sn+r)(4):=CHR$(i MOD 256)
  reprt(r)
  nb
ELIF t$="s" AND sn+r<=ne THEN
  cltp
  PRINT AT r,10: ">"
  INPUT AT 23,1: "sector? ": i
  e$(sn+r)(5):=CHR$(i MOD 256)
  reprt(r)
  nb
ELIF t$="p" THEN
  cltp
  PRINT AT 23,1: " printing..."
  pt
  nb
```

more»

```
    ENDIF
  UNTIL TRUE=FALSE
ENDPROC act
PROC pinc
  IF r=23 THEN
    r:=22; sn:+1
    IF sn>145-22 THEN sn:=145-22
    pag
  ENDIF
ENDPROC pinc
PROC help
  PRINT ""147"  Press f1 to insert a blank"
  PRINT " entry above present location."
  PRINT "   Press f3 for this help screen."
  PRINT "   Press f5 to move top of display"
  PRINT " up twelve entries."
  PRINT "   Press f7 to move top of display"
  PRINT " down twelve entries."
  PRINT "   Press f2 to delete entry."
  PRINT "   Press f4 to exit."
  PRINT "   Press f6 to display last entry."
  PRINT "   Press f8 to save revised directory."
  PRINT "   Press home twice to go to first"
  PRINT " entry."
  help2
ENDPROC help
PROC help2
  PRINT "   Press f to change filename."
  PRINT "   Press y to change file type listed."
  PRINT "(s=seq, S=protected seq, d=---, etc...)",
  PRINT "   Press b to change # blocks listed;"
  PRINT " t, track listed; s, sector listed."
  PRINT "   Press m to save a directory entry"
  PRINT " in temporary memory."
  PRINT "   Press r to recall all temporarily"
  PRINT " saved entries to above the present"
  PRINT " location."
  PRINT "   Press p to print directory in
  memory"//wrap line
  PRINT " on printer. (Press any key to
  return.)",//wrap line
ENDPROC help2
PROC pag
  PRINT CHR$(19),
  IF sn<0 THEN sn:=0
  IF sn>ne-20 AND ne>20 THEN sn:=ne-20
  wink
  FOR i:=1 TO 22 DO
    reprt(i)
  ENDFOR i
  on:=sn
ENDPROC pag
PROC kkey
  REPEAT
    k$:=KEY$
  UNTIL k$<>CHR$(0)
ENDPROC kkey
PROC cltp
  PRINT AT 23,1: SPC$(39),
ENDPROC cltp
FUNC typ$(thp$) CLOSED
  IF thp$ IN ""0""128""129""130""131""132""136""
  THEN //wrap line
    usl#:=1
  ELSE
    usl#:=0; j#:=ORD(thp$)
    i#:=j# BITAND 240
    thp$:=CHR$((j# MOD 32)+128)
  ENDIF
  IF thp$=""0"" THEN
    k$:=" --- "
  ELIF thp$=""128"" THEN
    k$:=" del "
  ELIF thp$=""129"" THEN
    k$:=" seq "
  ELIF thp$=""130"" THEN
    k$:=" prg "
  ELIF thp$=""131"" THEN
    k$:=" usr "
  ELIF thp$=""132"" THEN
    k$:=" rel "
  ELIF thp$=""136"" THEN
    k$:=" sr? "
  ELSE
    k$:=" ??? "
  ENDIF
  IF usl#=0 THEN
    IF i#=0 THEN
      k$(1):="*"
    ELIF i#=$c0 THEN
      k$(5):="<"
    ENDIF
```

more»

```
  ENDIF
  RETURN k$
ENDFUNC typ$
PROC intro
  PRINT ""147""17"  Press any key to begin
  loading"//wrap line
  PRINT " directory to be edited.",
  getk
ENDPROC intro
PROC pt
  OPEN FILE 7,"lp:",WRITE
  tn:=ne; i:=1
  PRINT FILE 7: TAB(11),dkname$,TAB(29),dkid$
  PRINT FILE 7: cr$
  WHILE tn>0 DO
    x$:=e$(i)(6:21)
    FOR j:=1 TO 16 DO
      IF w<32 OR (w>127 AND w<149) OR
      (w>155 AND w<160) THEN x$(j):=" "//wrap
    ENDFOR j
    PRINT FILE 7: TAB(2),ORD(e$(i)(31))+
    256*ORD(e$(i)(32)),//wrap line
    PRINT FILE 7: TAB(5),typ$(e$(i)(3)),
    PRINT FILE 7: TAB(11),x$,
    PRINT FILE 7: TAB(29),ORD(e$(i)(4)),
    TAB(31),"-",ORD(e$(i)(5)),//wrap line
    IF i<=ne THEN PRINT FILE 7: TAB(35),
    "18-",STR$(dir'sec((i-1) DIV 8)+3),//wrap
    tn:-1; i:+1
  ENDWHILE
  PRINT FILE 7: cr$
  CLOSE FILE 7
ENDPROC pt
PROC nb
  cltp //replace @ by "yellow" «ctrl»+8
  PRINT AT 23,1: "@ f3 help "18"",dkname$
  ," ",dkid$//wrap line
  PRINT "@ f1=insert entry f5=up 12   f6 last @
  "145""157""157""157""157""157""157""157""157"
  "157"f4 exit @"17""//double wrap line
  PRINT "@ f2=delete entry f7=down 12  f8
  save @",//wrap line
ENDPROC nb
PROC reprt(i)
  q:=sn+i; pname$(i):=e$(q)(6:21)
  IF e$(q)(33)<>z$ THEN
    FOR j:=1 TO 16 DO
      w:=ORD(pname$(i)(j))
      IF w=160 THEN
      ELIF w>31 AND w<128 THEN
      ELIF w>148 AND w<156 THEN
      ELIF w>160 THEN
      ELSE
        pname$(i)(j):=" "
      ENDIF
    ENDFOR j
  ENDIF
  PRINT AT i,1: SPC$(39),
  PRINT AT i,1: TAB(2),ORD(e$(q)(31))+
  256*ORD(e$(q)(32)),//wrap line
  PRINT TAB(5),typ$(e$(q)(3)),
  PRINT TAB(11),pname$(i),
  PRINT TAB(29),ORD(e$(q)(4)),TAB
  (31),"-",ORD(e$(q)(5)),//wrap line
  IF q<=ne THEN PRINT TAB(35),"18-",STR$
  (dir'sec((q-1) DIV 8)+3),//wrap line
ENDPROC reprt
PROC ee(ww)
  ii:=0
  FOR jj:=0 TO 15 DO
    w:=ORD(e$(ww)(6+jj))
    IF w=160 THEN
    ELIF w>31 AND w<128 THEN
    ELIF w>148 AND w<156 THEN
    ELIF w>160 THEN
    ELSE
      ii:+1
    ENDIF
  ENDFOR jj
  e$(ww)(33):=z$
  IF ii<>0 THEN e$(ww)(33):=CHR$(1)
ENDPROC ee
PROC wink
  IF pp<>0 THEN
    r:=pp-sn
    IF r<1 THEN r:=1
    IF r>22 THEN r:=22
    pp:=0
  ENDIF
ENDPROC wink
```

[see also Directory Designer in this issue] ■

ORDER FORM Subscriber#_____

Name:_____

Street:_____

City/St/Zip:_____

Visa/MC#:_____

Exp Date:_____ Signature:_____

Mar '88-Prices subject to change without notice

TO ORDER:

- Pull out this 4 page section
- Fill in subscriber# / address (above)
- Check [x] each item you want to order
- Add up total for items (fill in below)
- Add shipping/handling (fill in below)
- Send check/money order for Grand Total
  -- OR -- Fill in charge info (above) ...
  we will calculate the total & shipping
- Mail to: COMAL Users Group, U.S.A, Ltd,
  5501 Groveland, Madison, WI 53716
  ... or call 608-222-6625

SUBSCRIPTIONS:

Expired subscribers must renew before they may order at subscriber prices (renewal starts with the issue where you left off). New subscribers can order at the same time as subscribing.

[ ] $12.95 - 4 issue subscription/renewal
(Canada/APO add $1 per issue, 1st Class)
[ ] $29.95 - 4 disk subscription/renewal

[ ] $3.95 each **Early Issues:** 1 2 4
[ ] $3.95 each backissues; circle issues wanted:
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[ ] $16.95 *COMAL Yesterday* (spriral bound #1-4)

NOTICE: Minimum order $10. Shipping is extra: $3 minimum shipping; $3 per book; Canada, APO & 1st Class add $1 more per book and newsletter issue. Newsletter is published about 4 times per year; size and format may vary. Subscriptions may not extend more than 5 issues past the current issue. Cancelled subscriptions receive a credit; no money back. All orders must be prepaid in US Dollars, to Canada and USA only. Prices shown are subscriber prices and reflect a $2 discount. Allow 2 weeks for checks to clear. $10 charge for checks not honored by the bank. Wisconsin residents add 5% sales tax.

ENTER TOTALS HERE:

Item Total ($10 minimum) $_____
Ship Total ($3 minimum): $_____
Grand Total enclosed: $_____

SYSTEMS:

[ ] **C64 Power Box**[db]
Includes 3 utility disks, a toolbox of about 250 procedures and functions and a free compiler. $29.95 + $4 ship
[ ] **Starter Kit Option:** 12 issues of COMAL Today, 56 page index, 2 books and 5 disks! Over 1,000 pages! $29.95 + $4 ship
[ ] **C64 Keyboard Overlay Option:** excellent condensed command reference. $2.50
[ ] **Option** ... $6.95 Doc Box

[ ] **C64 COMAL 2.0 Cart Complete***
This is the way to go! It includes tutorials, references, packages, superchip on disk, etc. Cartridge with all 5 options: $168.95+$7 ship
[ ] **C64 COMAL 2.0 Cartridge***
64K cartridge with empty socket for up to 32K EPROM. Plain, no documentation $99.95
[ ] **Deluxe Option**[db]: three books: *2.0 Keywords*, *Cart Graphics & Sound*, *Common COMAL Reference*; 4 cart demo disks. $29.95+$4 ship
[ ] **Packages Option**[db]: three books: *COMAL 2.0 Packages*, *Packages Library 1* (17 packages), *Packages Library 2* (24 packages); Superchip on Disk (9 packages) $29.95 + $4 ship
[ ] **Super Chip Option:** plug in chip to add about 100 commands to the cartridge. $24.95
[ ] **Applications/Tutorial Option**[db]: three books: *COMAL Collage*, *3 Programs In Detail*, *Graph Paper*. $25.95 + $4 ship
[ ] **Option** ... $6.95 Doc Box

[ ] **CP/M COMAL 2.10**
Full COMAL system disk plus the DEMO disk, packed in a Doc Box with manual. Works in C128 CP/M mode. $49.95 + $4 ship
[ ] **Compiler Option:** RUNTIME system ... $5.95
[ ] **C128 Graphics Option:** Package disk $9.95
[ ] **CP/M Package Guide Option:** Reference on how to write packages $10.95 + $2 ship
[ ] **Option:** Common COMAL Reference $14.95

[ ] **UniComal IBM PC COMAL 2.1***
Full fast system, with extensive reference & tutorial packed in a Doc Box. $395 +$5 ship
[ ] **Compiler Option:** (PLUS version) adds a Runtime compiler and the Communication Package, with its own manuals packed in its own Doc Box. $195.00 + $4 ship
[ ] **Option:** Common COMAL Reference $14.95

[ ] **C128 COMAL 2.0 Cart***
Special order price: $195.95. This cart works on the C128 in its native mode. ($2 ship)
[ ] **Option**[db]: Common COMAL Reference $14.95
[ ] **Option** ... $6.95 Doc Box

db=Doc Box pages, requires a Doc Box for use.
*=subject to customs/ship variations/availability.

DISKS:

Take one free COMAL disk for each item over $5 you buy! Buy Beginning COMAL, get a free disk. Buy the Power Box, get a free disk. Renew your subscription, get a free disk. Even buy a disk, get another disk free! To buy a disk from the list, use a check mark or write the word buy in the [ ] box. The price is $7.95 per disk for subscribers. Free disks given only if requested by writing free in the box. Double sided disks (2 disks) count as 2 free disks, but still only cost $7.95! **Choose from the disks below:**

[ ] IBM Special Series Disk #1
[ ] CP/M COMAL Demo Disk ($5)
[ ] Beginning COMAL disk §
[ ] Foundations with COMAL disk §
[ ] COMAL Handbook disk §
[ ] COMAL Today INDEX disk § (2 disks)
[ ] Games Disk #1 (0.14 & 2.0)
[ ] Modem Disk (0.14 & 2.0)
[ ] Article text files disk

Today Disks:

[ ] Today Disk (one disk type--circle choices):
1 2 3 4 5
[ ] Today Disk (two disk type--circle choices):
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 22

0.14 Disks:

[ ] Data Base Disk 0.14
[ ] New: Power Driver Tutorial Disk
[ ] Auto Run Demo Disk
[ ] Paradise Disk
[ ] Best of COMAL
[ ] Bricks Tutorial (2 disks)
[ ] Utility Disk 1
[ ] Slide Show disk (circle which): 1 2
[ ] Spanish COMAL
[ ] User Group 0.14 disks (circle numbers):
1 2 3 4 5 6 7 8 9 10 12

2.0 Disks:

[ ] Data Base Disk 2.0
[ ] Superchip Programs disk
[ ] Read & Run
[ ] Math & Science
[ ] Typing disk (2 disks)
[ ] Cart Demo (circle which): 1 2 3 4
[ ] 2.0 user disks (circle choices): 11 13 14 15

§ = these disks assume you have the book

Note: Some disks may be supplied on the back side of another disk. Standard disk format is Commodore 1541 unless specified otherwise. We will replace any defective disk at no charge if you return the disk to us with a note explaining what is wrong with it.

SPECIAL DISKS & DISK SETS:

[ ] $10.95 **Sprite Pak**
Two disk set. Huge collection of sprite images, sprite editors, viewers, and other sprite programs. For 0.14 and 2.0.

[ ] $10.95 **Font Pak**
Three disk set. Collection of many different character sets (fonts) for use with 0.14 and 2.0 including special font editors!

[ ] $10.95 **Graphics Pak**
Five disk set. Picture heaven. Includes Slide Show, Picture Compactor, Graphics Editor and lots of pictures (normal and compacted)

[ ] $29.95 **Sprite, Font & Graphics Pak**
All ten disks mentioned above!

[ ] $9.95 **C128 CP/M Graphics**
Graphics package on disk for use with CP/M COMAL on the C128. Includes turtle graphics and preliminary Font package.

[ ] $10.95 **Guitar Disks**
Three 0.14 disk set. Teaches guitar by playing songs while displaying the chords and words.

[ ] $10.95 **Cart Demo Disks**
Four disks full of programs demonstrating the many features of the C64 2.0 cartridge.

[ ] $10.95 **Shareware Disks**
Three disk set. Includes a full HazMat system (Hazardous Materials), an Expert System, Finger Print system, Traffic Calc, and a BBS program.

[ ] $10.95 **Superchip On Disk**
All the commands of Super Chip (but not the Auto Start feature) disk loaded.

[ ] $10.95 **Super Chip Source Code**
Full source code with minimal comments. Customize your own Super Chip. Add commands. Remove the ones you don't need.

[ ] $10.95 **2.0 User Group Disks**
Four disks set for the C64 COMAL cart.

[ ] $24.95 **0.14 User Group Disks**
Twelve disks (User Group disk 9 is a newsletter on disk system, double disk).

[ ] $24.95 **European Disk Set**
Twelve 2.0 disk set. Find out what COMALites in Europe are doing.

BOOKS: (remember to add on the shipping fee)

[ ] $10.95 **Beginning COMAL**¤
#8 best seller by Borge Christensen
333 pages - General Textbook
Beginners text book, elementary school level, written by the founder of COMAL. This book is an easy reading text. You should find Borge has a good writing style, with a definite European flair.

[ ] $10.95 **Foundations with COMAL**¤
#4 best seller in Jan 88 by John Kelly
363 pages - General Textbook
Beginners text book, Jr/Sr High School level, including a section on C64 turtle graphics. A good text for those serious about programming.

[ ] $7.95 **COMAL Handbook**¤ (1st Edition)
by Len Lindsay - 2nd Edition is out of print
A few copies of 1st Edition are still left.
310 pages, it does not cover COMAL 2.0.
Common COMAL Reference has taken the place of the COMAL Handbook. See Below.

[ ] $3.95 **COMAL From A to Z**¤
#1 all time best seller by Borge Christensen
64 pages - Mini 0.14 Reference book
Written by the founder of COMAL

[ ] $3.95 **COMAL Workbook**¤
#1 best seller for Feb 88 by Gordon Shigley
69 pages - 0.14 Tutorial Workbook
Companion to the Tutorial Disk, great for beginners, full sized fill in the blank style.
[ ] **Tutorial Disk Option**: $7.95

[ ] $3.95 **Index to COMAL Today 1-12**
#9 best seller by Kevin Quiggle
52 page, 4,848 entry index to COMAL Today. The back issues of *COMAL Today* are a treasure trove of COMAL information and programs! This index is your key.
[ ] **Index Disk Option**: $7.95

[ ] $14.95 **Common COMAL Reference**db
#3 best seller for Feb 88 by Len Lindsay
238 page detailed cross reference to the COMAL implementations in the USA (formerly *COMAL Cross Reference*)
Covers: C64 COMAL 2.0, C128 COMAL 2.0, CP/M COMAL 2.10, Mytech IBM PC COMAL 2.0, and UniComal IBM PC COMAL 2.1

[ ] $2.95 **C64 COMAL 2.0 Keywords**db
#4 best seller for Feb 88 lists all the keywords built into the cartridge (including all 11 built-in packages) in alphabetic order complete with syntax and example.

[ ] $10.95 **CP/M COMAL Package Guide**db
The guide to making your own packages for CP/M COMAL by Richard Bain
76 pages - advanced package reference

[ ] $10.95 **Library Functions/Procedures**db
#1 best seller Dec 87 by Kevin Quiggle, 80 pages, over 100 0.14 procs/funcs, with disk

[ ] $4.95 **Cartridge Graphics & Sound**¤db
#6 best seller by Captain COMAL
64 pages - 2.0 packages reference
Reference guide to all the commands in the 11 built in cartridge packages.

[ ] $10.95 **COMAL 2.0 Packages**db
#7 all time best seller by Jesse Knight
108 pages with disk - package reference
How to write a package in Machine Code; includes C64 comsymb & supermon. For advanced users.

[ ] $10.95 **Package Library Vol 1**db
compiled by David Stidolph
76 pages with disk - package collection
17 packages ready to use, many with source code, plus the Smooth Scroll Editor!

[ ] $10.95 **Package Library Vol 2**db
67 pages with disk - package collection
24 example packages ready to use, most with source code, plus Disassembler, Re-Linker, De-Linker, Package Maker, Package Lister, and more. (includes windows)

[ ] $10.95 **COMAL Collage**db
by Frank and Melody Tymon
168 pages with disk, 2.0 programming guide, including graphics and sprites tutorial with many full sized example programs.

[ ] $10.95 **3 Programs in Detail**db
82 pages with disk by Doug Bittinger
Three 2.0 application programs explained: Blackbook (name/address system), Home Accountant, and BBS.

[ ] $10.95 **Graph Paper**db
52 pages with disk by Garrett Hughes
Function graphing system for COMAL 2.0. The program can't be LISTed. Includes a version for the Commodore Mouse.

[ ] $10.95 **COMAL Quick / Utility 2 & 3**db
#2 best seller Dec 87 by Jesse Knight
20 pages with 2 disks, fast loading COMAL 0.14, printer programs, utility programs.

db = Doc Box pages
¤ = while supplies last (out of print)

Playscore

by Jack Baldrige

One thing really confused me when I first got my COMAL 2.0 cartridge. I enjoyed the **playscore** programs on the demo disks, but I couldn't see how the programmers had got from the original song to the final program. Much, much later, I finally paid some concentrated attention to the playscore programs which I had. (I even made a rough translation of the Danish comments in one of them.) I found that the process isn't really very difficult, as is usually true when working with COMAL 2.0.

"*Write'playscore*" is a program which makes a disk data file from the data statements in a music program. The data statements which it reads give, for each note three alpha notes (one for each voice) and one number to indicate the duration of the note. There is no magic in the format I've just described. It happens to be the one I use for programming tunes. There are two variables in the program which are used to indicate duration. The variable **adsp#** governs the duration of the attack, sustain and decay part of the tone, and **rp#** the release part. These are incorporated in the the integers in the disk file. **Soundtype** and **adsr** will also be important in the final tune, although they are not included in the disk file.

"*Playscore'rose*" is a program to read the data file and play the tune from it. It reads the data file created by the "*write'playscore*" program. **Notice**: when the program is run, the song is played automatically in the background, and will continue playing even if the program is stopped. This is automated music. Use the command: **volume(0)** or **discard** to turn off the music.

If you would like to find for yourself how simply a playscore file can be made and used, take your *Today Disk #22* (or type in the programs listed after this article to create your own disk). Now try this:

- load the program "*auld'lang'syne*"
- list its data statements to disk:
  LIST xxx- "name"
  replace xxx by the line# of the first data line
- load "write'playscore"
- delete the data statements at the end of the program (they are for the *rose* song):
  DEL xxx-
  replace xxx by the line# of the first data line
- merge the "auld'lang'syne" data from the disk file you just made in step 2:
  MERGE "name"

You may have to play the tune several times, while varying soundtype, adsr, and the two duration variables, before you make a disk file. You may then play the tune from using the file you create with the program "*playscore'rose*" after changing the file name, soundtype and adsr in it.

Why do it? If you examine Colin Thompson's program "*all'at'once*", on cartridge demo disk #2, you'll see how many things you can do while playing a tune, such as drawing a design and animating a sprite. In his program, however, he used a playscore file created by UniComal. Now, you may also use a tune which you have programmed by using "*write'playscore*". I plan to try it some more. How about you?

Program to create a playscore data file

```
// delete"write'playscore
//   save"write'playscore
// Write playscore disk file
// by Jack Baldrige, Boulder CO
USE sound
DIM freq1#(100), adsp1#(100), rp1#(100)
DIM freq2#(100), adsp2#(100), rp2#(100)
DIM freq3#(100), adsp3#(100), rp3#(100)
DIM a$ OF 3, b$ OF 3, c$ OF 3
DIM name$ OF 16
ct1#:=1; ct2#:=1; ct3#:=1; tm#:=10
PAGE
PRINT " Reading data statements"
```

more»

```
FOR k#:=1 TO 3 DO soundtype(k#,2);
adsr(k#,0,4,10,10)
WHILE NOT EOD DO
  READ a$,b$,c$,d
  set'voices(ct1#,a$,d,freq1#(),
  adsp1#(),rp1#(),tm#)//wrap line
  set'voices(ct2#,b$,d,freq2#(),
  adsp2#(),rp2#(),tm#)//wrap line
  set'voices(ct3#,c$,d,freq3#(),
  adsp3#(),rp3#(),tm#)//wrap line
  ct1#:+1; ct2#:+1; ct3#:+1
ENDWHILE
//
PROC set'voices(REF ct#,x$,d,REF freq#(),
REF adsp#(),REF rp#(),tm#) //wrap line
  IF x$="r" THEN
    ct#:-1; rp#(ct#):+d/25+tm#
  ELIF x$="*" THEN
    ct#:-1; adsp#(ct#):+d/25+tm#
  ELSE
    freq#(ct#):=frequency(x$)
    adsp#(ct#):=d/25; rp#(ct#):=tm#
  ENDIF
ENDPROC set'voices
//
INPUT ""17" Write disk file? N"157"": q$
IF q$="y" OR q$="Y" THEN
  PRINT ""17" Filename: dat.rose",
  INPUT AT 0,12,16: "": name$
  PRINT ""17" Writing file:";name$
  write'file(name$)
ENDIF
PRINT ""17" Playing ""Es Ist Ein' Ros""""
setscore(1,freq1#(),adsp1#(),rp1#())
setscore(2,freq2#(),adsp2#(),rp2#())
setscore(3,freq3#(),adsp3#(),rp3#())
playscore(1,1,1)
WHILE NOT waitscore(1,1,1) DO NULL
volume(0)
END "Done"
//
PROC write'file(name$)
  TRAP
    OPEN FILE 2,name$,WRITE
    WRITE FILE 2: ct1#
    WRITE FILE 2: ct2#
    WRITE FILE 2: ct3#
    FOR k:=1 TO ct1# DO WRITE FILE
    2: freq1#(k)//wrap line
    FOR k:=1 TO ct1# DO WRITE FILE
    2: adsp1#(k)//wrap line
    FOR k:=1 TO ct1# DO WRITE FILE
    2: rp1#(k)//wrap line
    FOR k:=1 TO ct2# DO WRITE FILE
    2: freq2#(k)//wrap line
    FOR k:=1 TO ct2# DO WRITE FILE
    2: adsp2#(k)//wrap line
    FOR k:=1 TO ct2# DO WRITE FILE
    2: rp2#(k)//wrap line
    FOR k:=1 TO ct3# DO WRITE FILE
    2: freq3#(k)//wrap line
    FOR k:=1 TO ct3# DO WRITE FILE
    2: adsp3#(k)//wrap line
    FOR k:=1 TO ct3# DO WRITE FILE
    2: rp3#(k)//wrap line
    CLOSE FILE 2
  HANDLER
    PRINT "Disk error-file aborted"
    CLOSE FILE 2
  ENDTRAP
ENDPROC write'file
//
DATA "c5","a4","f3",600
DATA "c5","a4","f3",300
DATA "c5","f4","f3",300
DATA "d5","f4","a3#",300
DATA "c5","f4","f3",300
DATA "c5","e4","c3",600
DATA "a4","d4","d3",600
DATA "a4#","d4","a2#",600
DATA "a4","c4","f3",300
DATA "g4","*","e3",300
DATA "*","d4","d3",300
DATA "f4","d4","d3",150
DATA "*","a3","*",150
DATA "*","c4","c3",300
DATA "e4","*","*",300
DATA "f4","c4","f2",600
DATA "r","r","r",100
DATA "c5","a4","f3",600
DATA "c5","a4","f3",300
DATA "c5","f4","f3",300
```

more»

```
DATA "d5","f4","a3#",300
DATA "c5","f4","f3",300
DATA "c5","e4","c3",600
DATA "a4","d4","d3",600
DATA "a4#","d4","a2#",600
DATA "a4","c4","f3",300
DATA "g4","*","e3",300
DATA "*","d4","d3",300
DATA "f4","d4","d3",150
DATA "*","a3","*",150
DATA "*","c4","c3",300
DATA "e4","*","*",300
DATA "f4","c4","f2",600
DATA "r","r","r",400
DATA "a4","r","r",300
DATA "g4","r","r",300
DATA "e4","r","r",300
DATA "f4","c4","r",300
DATA "d4","b3","r",300
DATA "c4","c4","r",150
DATA "*","d4","r",150
DATA "*","e4","r",600
DATA "r","r","r",100
DATA "c5","e4","c3",300
DATA "c5","g4","e3",300
DATA "c5","f4","f3",300
DATA "d5","f4","a2#",300
DATA "c5","f4","f2",300
DATA "c5","e4","c3",600
DATA "a4","d4","d3",600
DATA "a4#","d4","g2",600
DATA "a4","f4","a2",300
DATA "g4","d4","a2#",300
DATA "*","e4","c3",300
DATA "f4","f4","c3",300
DATA "*","g4","c2",300
DATA "e4","c4","*",300
DATA "f4","c4","f2",720
DATA "*","*","*",720
```

Program to play a tune using data file

```
// delete"playscore'rose
//   save"playscore'rose
// by Jack Baldrige, Boulder CO
USE sound
FOR k#:=1 TO 3 DO soundtype(k#,2);
adsr(k#,0,4,10,10)
name$:="dat.rose"
PAGE
PRINT " Reading file:";name$
OPEN FILE 2,"dat.rose",READ
READ FILE 2: ct1#
READ FILE 2: ct2#
READ FILE 2: ct3#
DIM f1#(ct1#), p11#(ct1#), p21#(ct1#)
DIM f2#(ct2#), p12#(ct2#), p22#(ct2#)
DIM f3#(ct3#), p13#(ct3#), p23#(ct3#)
READ FILE 2: f1#(),p11#(),p21#()
READ FILE 2: f2#(),p12#(),p22#()
READ FILE 2: f3#(),p13#(),p23#()
CLOSE FILE 2
PRINT ""17" Playing ""Es Ist Ein' Ros'""""
setscore(1,f1#(),p11#(),p21#())
setscore(2,f2#(),p12#(),p22#())
setscore(3,f3#(),p13#(),p23#())
volume(15)
playscore(1,1,1)
WHILE NOT waitscore(1,1,1) DO NULL
volume(0)
END "Done"
```

Program to play a tune with own data

```
// delete"0:auld'lang'syne
//   save"0:auld'lang'syne
// Jack Baldrige, Boulder CO
USE sound
PAGE
PRINT AT 2,2: "Playing ""Auld Lang Syne"""
DIM a$ OF 3, b$ OF 3, c$ OF 3
FOR k#:=1 TO 3 DO soundtype(k#,2);
adsr(k#,1,4,10,8)
volume(15)
play
gate(1,0); gate(2,0); gate(3,0)
volume(0)
END "Done"
//
PROC play
  LOOP
```

more»

```
    READ a$,b$,c$,d
    IF a$<>"*" THEN gate(1,0)
    IF b$<>"*" THEN gate(2,0)
    IF c$<>"*" THEN gate(3,0)
    FOR i:=1 TO 50 DO NULL
    IF a$<>"*" AND a$<>"r" THEN note(1,a$);
    gate(1,1)//wrap line
    IF b$<>"*" AND b$<>"r" THEN note(2,b$);
    gate(2,1)//wrap line
    IF c$<>"*" AND c$<>"r" THEN note(3,c$);
    gate(3,1)//wrap line
    FOR i:=1 TO d DO NULL
    EXIT WHEN EOD
  ENDLOOP
ENDPROC play
//
DATA "a3#","r","r",250
DATA "d4#","a3#","d2#",375
DATA "d4#","*","*",125
DATA "d4#","a3#","d2#",250
DATA "g4","*","*",250
DATA "f4","g3#","g2#",375
DATA "d4#","*","*",125
DATA "f4","g3#","g2#",250
DATA "g4","*","*",250
DATA "d4#","g3","d2#",375
DATA "d4#","*","*",250
DATA "g4","g3","d2#",250
DATA "a4#","a3#","g2",250
DATA "c5","d4#","g2#",250
DATA "*","*","a2#",250
DATA "*","*","c3",250
DATA "c5","d4#","g2#",250
DATA "a4#","a3#","g2",375
DATA "g4","a3#","d2#",125
DATA "g4","a3#","d2#",125
DATA "d4#","a3#","g2",250
DATA "f4","d4","a2#",375
DATA "d4#","c4","*",125
DATA "f4","a3#","a2#",250
DATA "g4","g3","b2",250
DATA "d4#","c4","c3",375
DATA "c4","c4","r",125
DATA "c4","g3#","f2",250
DATA "a3#","g3#","a2#",250
DATA "d4#","a3#","g3",250
DATA "*","*","a2#",250
DATA "*","*","d2#",250
DATA "c5","d4#","g3#",250
DATA "a4#","d4#","g3",375
DATA "g4","a3#","d3#",125
DATA "g4","a3#","d3#",250
DATA "d4#","a3#","g3",250
DATA "f4","d4","a3#",375
DATA "d4#","c4","a3#",125
DATA "f4","d4","a3#",250
DATA "c5","d4#","g3#",250
DATA "a4#","d4#","g3",375
DATA "g4","a3#","d3#",125
DATA "g4","a3#","d3#",250
DATA "a4#","c4#","d3#",250
DATA "c5","c4","g2#",250
DATA "*","d4","a2#",250
DATA "*","d4#","c3",250
DATA "d5#","c4","g2#",250
DATA "a4#","d4#","g2",375
DATA "g4","a3#","d2#",125
DATA "g4","*","*",250
DATA "d4#","a3#","g2",250
DATA "f4","d4","a2#",375
DATA "d4#","c4","a2#",125
DATA "f4","d4","a2#",250
DATA "g4","b3","g2",350
DATA "d4#","g3","c3",175
DATA "*","*","a2#",175
DATA "*","*","g2#",175
DATA "c4","c4","g2",200
DATA "c4","g3#","f2",500
DATA "a3#","g3#","a2#",600
DATA "d4#","a3#","d2#",1600
```

Further Reference:

Music To Your Ears, David Stidolph, #4 pg 40
Song Editor 2.0, Mark Clifford, #6 pg 68
Music Writer 0.14, David Stidolph, #8 pg 30
Sound Routines 0.14, David Stidolph, #8 pg 34
Sounds From Holland, #8 pg 35
Tanks & Animate, Bob Hoerter, #9 pg 64
Sound Effects, #12 pg 61
Sound Keywords, #12 pg 55 ■

Music

by Jean Nance

"*Music*" allows you to enter music chords into the program. At any time you can see a graphic display showing the designations for the notes on a staff. At any point after entering one or more chords, you may listen to the chords, print to the screen the code entered, change the notes or duration of notes for any chord, or see the code you have entered displayed on the screen. You may then continue entering music, or save the musical piece to disk.

There are two versions of "*music*", one for COMAL 0.14 and one for cartridge COMAL 2.0. The cartridge version gives a choice of violin, piano, or organ sound. The COMAL 0.14 version does not offer this option. There is a limit on the number of chords that can be entered: 300 for cartridge version, 108 for COMAL 0.14.

The program was written first in COMAL 2.0, using the 2.0 music commands. For the 0.14 version, procedures from "*Sound Routines for 0.14*" by David Stidolph, *COMAL Today #8* were used to simulate COMAL 2.0 music commands.

Two music files are included on *Today Disk #22*, "*A Mighty Fortress is our God*", by Martin Luther, and "*Ode to Joy*", by Beethoven, to show what is possible with the program.

COMAL 0.14 Program Listing

```
setup
start
directions
c:=1
input'notes
end
//
proc start
 print "PLAY A TUNE WITH COMAL"
 print
 print "hit 'n' to write a (n)ew tune"
 print "hit 'l' to (l)oad a tune from disk"
 input ch$
 print
 if ch$="l" then load'music
endproc start
//
proc setup
 dim n$ of 3, notes$ of 108, note$(2) of 57
 dim answ$ of 1, ans$ of 1, ch$ of 1
 dim frequency(36), code$ of 3
 dim dur(1:100), dr$(1:100) of 2
 dim nte$(1:100,1:3) of 3
 dim filename$ of 16, new'note$ of 3
 background 5
 pencolor 1
 poke 54296,15
 soundtype(1,1)
 soundtype(2,1)
 soundtype(3,1)
 note$(1):="a2 a2#b2 c3 c3#d3 d3#e3 e3#f3 g3
  g3#a3 a3#b3 c4 c4#"//wrap line
 note$(2):="d4 d4#e4 f4 f4#g4 g4#a4a4#b4 c5
  c5#d5 d5#e5 f5 f5#g5 z  "//wrap line
 notes$:=note$(1)+note$(2)
 for x:=1 to 36 do read frequency(x)
 adsr(1,10,8,10,9)
 adsr(2,10,8,10,9)
 adsr(3,10,8,10,9)
 print chr$(147)
endproc setup
//
proc directions
 print "     Hit 'm' to see music notes"
 print "     Hit any other key to start"
 input ans$
 if ans$="m" then
  v:=0
  music'screen
 else
  intro
 endif
endproc directions
//
proc intro
 print chr$(147)
 print "Enter chords of three notes each"
 print "Example:"
```

more»

```
 print "Chord 1 Voice 1  c5"
 print "Chord 1 Voice 2  e4"
 print "Chord 1 Voice 3  c4"
 print
 print "Use '#' after a note to make it a sharp"
 print "Remember, there is no e# or b#"
 print
 print "Type 'z' for a pause."
 print "Type 'm' to see music screen      "
 print
 print "If you make certain errors"
 print "You will need to enter that chord again"
 print "You may correct notes later,too"
 print
 print "108 chords maximum"
 print
 print "Enter duration for each chord"
 print "1 for whole note to 32 for 1/32 note"
 print "Type '0' to end."
 print
endproc intro
//
proc input'notes
 for v:=1 to 3 do
  print "Chord";c;" ";"Voice";v;
  input n$
  if n$="m" then music'screen
  if n$="0" then play'tune
  if len(n$)=1 and n$<>"m" and n$<>"z" then
   print "error- at least 2 characters"
   input'notes
  endif
  if len(n$)>1 then
   if n$(1)<"a" or n$(1)>"g" and n$<>"z" and
   n$<>"m" then//wrap line
    print "error- must be 'a' to 'g'"
    input'notes
   endif
  endif
  if len(n$)=2 then
   if n$(2)<"2" or n$(2)>"5" then
    print "error- 2nd character must be 2,3,4,or
    5"//wrap line
    input'notes
   endif
  endif
  if len(n$)=3 then
   if n$(3)<>"#" then
    print "error- must be # or nothing"
    input'notes
   endif
  endif
  if len(n$)=3 then
   if n$(1)="e" or n$(1)="b" and n$(3)="#" then
    print "no such note"
    input'notes
   endif
  endif
  if n$<>"m" then
   nte$(c,v):=n$
  endif
 endfor v
 print
 input'duration
 for t:=1 to 250 do null
 c:=c+1
 print
 input'notes
endproc input'notes
//
proc play'tune
 sid:=54272
 for x:=1 to c-1 do
  for v:=1 to 3 do
   j:=((nte$(x,v) in notes$)+2)/3
   gate(v,1)
   poke sid+(v-1)*7,frequency(j) mod 256
   poke sid+1+(v-1)*7,frequency(j) div 256
  endfor v
  for t:=1 to 512/dur(x) do null
  gate(1,0)
  gate(2,0)
  gate(3,0)
 endfor x
 choice
endproc play'tune
//
proc choice
 print
 print "a to play (a)gain."
 print "m to enter (m)ore chords."
 print "c to (c)orrect notes."
```

more»

```
 print "p to see notes (p)rinted."
 print "s to (s)ave tune to disk and stop"
 print "l to (l)oad another tune from disk"
 print "q  to (q)uit."
 print
 input ch$
 case ch$ of
 when "a"
  play'tune
 when "m"
  input'notes
 when "c"
  correct
 when "s"
  save'music
 when "p"
  print'again
 when "l"
  load'music
 when "q"
  end
 otherwise
  null
 endcase
endproc choice
//
proc music'screen
 setgraphic 0
 hideturtle
 clear
 background 5
 pencolor 1
 for line:=180 to 100 step -20 do
  moveto 0,line
  drawto 319,line
 endfor line
 moveto 230,80
 drawto 319,80
 for line:=60 to 20 step -20 do
  moveto 0,line
  drawto 319,line
 endfor line
 plottext 260,80,"c4"
 plottext 200,90,"e4"
 plottext 200,110,"f4"
 plottext 260,160,"d5"
 plottext 200,168,"e5"
 plottext 260,180,"f5"
 plottext 260,120,"g4"
 plottext 200,150,"c5"
 plottext 260,140,"b4"
 plottext 200,128,"a4"
 plottext 200,30,"e3"
 plottext 200,90,"d4"
 plottext 260,100,"e4"
 plottext 200,70,"b3"
 plottext 260,60,"a3"
 plottext 200,48,"g3"
 plottext 260,40,"f3"
 plottext 260,20,"d3"
 plottext 10,168,"hit f1 to return"
 plottext 200,8,"c3"
 plottext 10,148,"to program"
 while key$<>chr$(133) do null
 if v=0 then
  intro
  settext
 endif
 v:=v-1
 if v=0 then
  v:=3
  c:=c-1
 endif
 settext
 poke 53272,23
endproc music'screen
//
proc correct
 input "correct (n)ote or (d)uration? ": answ$
 if answ$="d" then correct'duration
 print "note number to correct,voice,new note"
 input num,voice,new'note$
 nte$(num,voice):=new'note$
 input "Another correction (y/n) ": answ$
 if answ$="y" then correct
 if answ$="n" then choice
endproc correct
//
proc correct'duration
 input "number of chord,new duration ":
 ch'num,dur'correct//wrap line
 dur(ch'num):=dur'correct
```

more»

```
 input "another correction (y/n) ": answ$
 if answ$="y" then correct
 if answ$="n" then choice
endproc correct'duration
//
proc print'again
 for x:=1 to c-1 do
  for v:=1 to 3 do
   print "Note";x;" ";
   print "voice";v;"  ";
   print nte$(x,v)
  endfor v
  print "duration: 1/";dur(x)
  if x/5=int(x/5) then pause2
 endfor x
 choice
endproc print'again
//
proc pause2
 print "Hit any key for more"
 while key$=chr$(0) do null
endproc pause2
//
proc save'music
 input "filename ": filename$
 if file'exists(filename$) then
  print
  print "File by that name on disk"
  print
  save'music
 endif
 open file 1,filename$,write
 print file 1: c
 for x:=1 to c-1 do
  for v:=1 to 3 do print file 1: nte$(x,v)
  print file 1: dur(x)
 endfor x
 close file 1
 end
endproc save'music
//
proc load'music
 input "file name ": filename$
 open file 1,filename$,read
 input file 1: c
 for x:=1 to c-1 do
  for v:=1 to 3 do input file 1: nte$(x,v)
  input file 1: dur(x)
 endfor x
 close file 1
 play'tune
endproc load'music
//
proc soundtype(v,t) closed
 temp:=peek(1020+v)
 temp:=temp mod 16
 if t>0 then temp:+((2*t)-1)*16
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc soundtype
//
func val(x$) closed
 open file 100,"#",unit 8,2,read
 print file 100: x$
 pass "b-p:2,1"
 input file 100: y
 close file 100
 return y
endfunc val
//
proc adsr(v,a,d,s,r)
 poke 54277+(v-1)*7,a*16+d
 poke 54278+(v-1)*7,s*16+r
endproc adsr
//
proc gate(v,on'off)
 temp:=int(peek(1020+v)/2)*2
 if on'off then temp:+1
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc gate
//
data 1804,1911,2025,2145,2273,2408,2551,2703,2864
data 3034,3215,3406,3608,3823,4050,4291,4547,4817
data 5103,5407,5728,6069,6430,6812,7217,7647,8101
data 8583,9094,9634,10207,10814,11457,12139,12860
data 0
//
proc input'duration
 input "duration (1,2,4,8,16,32) ": dr$(c)
 if dr$(c) in "1 2 4 8 1632" and dr$(c)<>"  "
 and dr$(c)<>"" then//wrap line
```

more»

```
  dur(c):=val(dr$(c))
 else
  print
  print "Duration must be 1,2,4,8,16 or 32"
  print
  input'duration
 endif
endproc input'duration
//
func file'exists(filename$) closed
 dim s$ of 2
 open file 78,filename$,read
 s$:=status$
 close file 78
 if s$="62" then
  return false
 else
  return true
 endif
endfunc file'exists
```

C64 2.0 Cartridge Listing

```
//Music program
setup
start
choose'sounds
directions
c:=1
input'notes
END
//
PROC start
  PRINT "PLAY A TUNE WITH COMAL"
  PRINT
  PRINT "hit 'n' to write a (n)ew tune"
  PRINT "hit 'l' to (l)oad a tune from disk"
  INPUT ch$
  PRINT
  IF ch$="l" THEN load'music
ENDPROC start
//
PROC setup
  DIM n$ OF 3, ch$ OF 1, code$ OF 3
  DIM nte$(1:300,1:3) OF 3, dur(1:300)
  USE system
  USE sound
  soundtype(1,2)
  soundtype(2,2)
  soundtype(3,2)
  textcolors(5,5,1)
  PAGE
ENDPROC setup
//
PROC choose'sounds
  PRINT "(v)iolin, (p)iano, or (o)rgan sound"
  INPUT snd$
  IF snd$="v" THEN
    adsr(1,10,8,10,9)
    adsr(2,10,8,10,9)
    adsr(3,10,8,10,9)
  ELIF snd$="p" THEN
    adsr(1,0,9,0,0)
    adsr(2,0,9,0,0)
    adsr(3,0,9,0,0)
  ELIF snd$="o" THEN
    adsr(1,4,4,12,10)
    adsr(2,4,4,12,10)
    adsr(3,4,4,12,10)
  ELSE
    PRINT
    PRINT "Wrong letter "
    choose'sounds
  ENDIF
ENDPROC choose'sounds
//
PROC directions
  PRINT "     Hit 'm' to see music notes"
  PRINT "     Hit any other key to start"
  INPUT ans$
  IF ans$="m" THEN
    v:=0
    music'screen
  ELSE
    intro
  ENDIF
ENDPROC directions
//
PROC intro
  PAGE
  PRINT "Enter chords of three notes each"
  PRINT "Example:"
```

more»

```
  PRINT "Chord 1 Voice 1  c5"
  PRINT "Chord 1 Voice 2  e4"
  PRINT "Chord 1 Voice 3  c4"
  PRINT
  PRINT "Use # after a note to make it sharp"
  PRINT "Remember, there is no e# or b#"
  PRINT
  PRINT "Type 'z' for a pause."
  PRINT "type 'm' to see music screen     "
  PRINT
  PRINT "If you make certain errors"
  PRINT "You'll need to enter that chord again"
  PRINT "You may correct notes later, too"
  PRINT
  PRINT "300 chords maximum"
  PRINT
  PRINT "enter duration for each chord"
  PRINT "1 for whole note to 32 for 1/32 note"
  PRINT "Type '0' to end."
ENDPROC intro
//
PROC input'notes
  FOR v:=1 TO 3 DO
    PRINT "Chord";c;" ";"Voice";v;
    INPUT AT 0,0,3: n$
    IF n$="m" THEN music'screen
    IF n$="0" THEN play'tune
    IF len(n$)=1 and n$<>"m" and n$<>"z" THEN
      PRINT "error- at least 2 characters"
      input'notes
    ENDIF
    IF LEN(n$)>1 THEN
      IF n$(1)<"a" OR n$(1)>"g" AND
      n$<>"z" AND n$<>"m" THEN//wrap line
        PRINT "error- must be 'a' to 'g'"
        input'notes
      ENDIF
    ENDIF
    IF LEN(n$)=2 THEN
      IF n$(2)<"2" OR n$(2)>"5" THEN
        PRINT "error- 2nd character must be
        2,3,4,or 5"//wrap line
        input'notes
      ENDIF
    ENDIF
    IF LEN(n$)=3 THEN
      IF n$(3)<>"#" THEN
        PRINT "error- must be # or nothing"
        input'notes
      ENDIF
    ENDIF
    IF LEN(n$)=3 THEN
      IF n$(1)="e" OR n$(1)="b" AND
      n$(3)="#" THEN//wrap line
        PRINT "no such note"
        input'notes
      ENDIF
    ENDIF
    IF n$<>"m" THEN
      nte$(c,v):=n$
    ENDIF
  ENDFOR v
  PRINT
  LOOP
    TRAP
      INPUT "duration (1,2,4,8,16,32) ": dur(c)
      IF dur(c)=0 THEN
        PRINT
        PRINT "Duration must be
        1,2,4,8,16 or 32"//wrap line
        PRINT
        INPUT "Duration (1,2,4,8,16,32) ": dur(c)
      ENDIF
      PRINT
      EXIT
    HANDLER
      PRINT
      PRINT "Duration must be 1,2,4,8,16,or 32!"
      PRINT
      FOR pause:=1 TO 1000 DO NULL
    ENDTRAP
  ENDLOOP
  c:=c+1
  input'notes
ENDPROC input'notes
//
PROC play'tune
  FOR x:=1 TO c-1 DO
    FOR v:=1 TO 3 DO
      IF nte$(x,v)<>"z" THEN
        note(v,nte$(x,v))
        gate(v,1)
```

more»

```
      ENDIF
    ENDFOR v
    pause(25/dur(x))
    gate(1,0)
    gate(2,0)
    gate(3,0)
    FOR t:=1 TO 140 DO NULL
  ENDFOR x
  choice
ENDPROC play'tune
//
PROC choice
  PRINT
  PRINT "a to play (a)gain."
  PRINT "m to enter (m)ore chords."
  PRINT "c to (c)orrect notes."
  PRINT "p to see notes (p)rinted."
  PRINT "s to (s)ave tune to disk and stop"
  PRINT "l to (l)oad another tune from disk"
  PRINT "q  to (q)uit."
  INPUT ch$
  CASE ch$ OF
  WHEN "a"
    play'tune
  WHEN "m"
    input'notes
  WHEN "c"
    correct
  WHEN "q"
    STOP
  WHEN "p"
    print'again
  WHEN "s"
    save'music
  WHEN "l"
    load'music
  OTHERWISE
    NULL
  ENDCASE
ENDPROC choice
//
PROC music'screen
  USE graphics
  USE turtle
  hideturtle
  graphicscreen(0)
  background(5)
  pencolor(1)
  FOR line:=180 TO 100 STEP -20 DO
    moveto(0,line)
    drawto(319,line)
  ENDFOR line
  moveto(230,80)
  drawto(319,80)
  FOR line:=60 TO 20 STEP -20 DO
    moveto(0,line)
    drawto(319,line)
  ENDFOR line
  plottext(260,80,"C4")
  plottext(260,180,"F5")
  plottext(200,168,"E5")
  plottext(260,160,"D5")
  plottext(200,148,"C5")
  plottext(260,140,"B4")
  plottext(200,128,"A4")
  plottext(260,120,"G4")
  plottext(200,108,"F4")
  plottext(260,100,"E4")
  plottext(200,88,"D4")
  plottext(200,68,"B3")
  plottext(260,60,"A3")
  plottext(200,48,"G3")
  plottext(260,40,"F3")
  plottext(200,28,"E3")
  plottext(260,20,"D3")
  plottext(10,168,"Hit F1 to return")
  plottext(200,8,"C3")
  plottext(10,148,"to program")
  WHILE KEY$<>CHR$(133) DO
    graphicscreen(0)
  ENDWHILE
  IF v=0 THEN
    intro
    textscreen
  ENDIF
  v:=v-1
  IF v=0 THEN
    v:=3
    c:=c-1
  ENDIF
  textscreen
ENDPROC music'screen
```

more»

```
//
PROC play(voice,code$)
  IF code$<>"z" THEN note(voice,code$)
ENDPROC play
//
PROC pause(sec'32)
  TIME 0
  WHILE TIME<1.875*sec'32 DO NULL
ENDPROC pause
//
PROC correct
  INPUT "correct (n)ote or (d)uration? ": answ$
  IF answ$="d" THEN correct'duration
  PRINT "note number to correct,voice,new
  note"//wrap line
  INPUT num,voice,new'note$
  nte$(num,voice):=new'note$
  INPUT "Another correction (y/n) ": answ$
  IF answ$="y" THEN correct
  IF answ$="n" THEN choice
ENDPROC correct
//
PROC correct'duration
  INPUT "number of chord,new duration ":
  ch'num,dur'correct//wrap line
  dur(ch'num):=dur'correct
  INPUT "another correction (y/n) ": answ$
  IF answ$="y" THEN correct
  IF answ$="n" THEN choice
ENDPROC correct'duration
//
PROC print'again
  FOR x:=1 TO c-1 DO
    FOR v:=1 TO 3 DO
      PRINT "Note";x;" ";
      PRINT "voice";v;"  ";
      PRINT nte$(x,v)
    ENDFOR v
    PRINT "duration: 1/";dur(x)
    IF x/5=INT(x/5) THEN pause2
  ENDFOR x
  choice
ENDPROC print'again
//
PROC pause2
  PRINT "Hit any key for more"
  WHILE KEY$=CHR$(0) DO NULL
ENDPROC pause2
//
PROC save'music
  TRAP
    INPUT "filename ": filename$
    OPEN FILE 1,filename$,WRITE
    PRINT FILE 1: c
    FOR x:=1 TO c-1 DO
      FOR v:=1 TO 3 DO
        PRINT FILE 1: nte$(x,v)
      ENDFOR v
      PRINT FILE 1: dur(x)
    ENDFOR x
    CLOSE FILE 1
    END
  HANDLER
    IF ERR=274 THEN
      PRINT
      PRINT "disk drive not ready!"
      PRINT
      save'music
    ELIF ERR=263 THEN
      PRINT
      PRINT "File by that name exists!"
      PRINT
      save'music
    ENDIF
  ENDTRAP
  END
ENDPROC save'music
//
PROC load'music
  INPUT "file name ": filename$
  OPEN FILE 1,filename$,READ
  INPUT FILE 1: c
  FOR x:=1 TO c-1 DO
    FOR v:=1 TO 3 DO INPUT FILE 1: nte$(x,v)
    INPUT FILE 1: dur(x)
  ENDFOR x
  CLOSE FILE 1
  choose'sounds
  play'tune
ENDPROC load'music ■
```

Multident

by Bill Inhelder

To play some games well requires a high degree of hand-eye coordination. Such games can be very frustrating to those of us whose reflexes have slowed down due to age. Personally, I prefer games which are not time dependent and which require analysis and the development of a strategy to play well. Multident is such a game.

Letters **a,b,c,d,e,f,g,h,i,j** are paired randomly with the numbers **0,1,2,3,4,5,6,7,8,9.** The object of the game is to determine which number is paired with each letter.

In determining the pairing, you may ask for the sum, difference or product of any two letters and the program will supply the numerical answer in the form of its paired letter. Thus, for example, you may ask: c+f or **a-d** or **b*g**. If the solution is greater than two digits, the program will supply a letter which stands for the unit's digit. For subtraction, a negative answer will be given in its positive form.

When you think you know the value of a letter, type it in this form: **c=6.** The program will tell you if it is correct. The number of inquiries of both types will be recorded. Once you have identified them all correctly, your game will be evaluated on the basis of the number of inquiries made. To achieve a rating of excellent, you must not exceed 19 inquiries. This rating is not easy to achieve without an efficient strategy and luck since 10 inquiries are needed to identify the 10 numbers. Paper and pencil can be helpful in solving the puzzle.

If you prefer to develop your own strategy for playing the game, do not read any further.

Some basic number properties are significant in playing this game efficiently. The first property states that a number subtracted from itself will always produce a zero (called the identity element for addition). The second property, which is of crucial importance to the game, states that if **x*y=x** then y must equal 1 (the identity element for multiplication from which the name of the game is taken). However, we must be careful because of the way the product of two numbers is represented in the program. Remember only the unit's digit of the product is displayed! Unfortunately such products as: **4*6=4, 5*7=5, 2*6=2,** etc. might lead you astray. However, once the number 1 is identified, then 2 is easily determined, then 3, etc.

An alternate strategy involves the square of a number. After zero has been identified, there are only 3 numbers whose square results in a product with the same number in the unit's position (**1*1=1, 5*5=5, 6*6=6**). This provides one chance in three of correctly identifying the identity element for multiplication. Even with the best strategy there is an element of luck in attaining an excellent rating.

```
PAGE // for Power Driver & 2.0
DIM a$(0:9) OF 1, c$(0:9) OF 1, d(0:10)
DIM b$ OF 40, h$ OF 40, ans$ OF 1, cc$ OF 1
heading
instructions
REPEAT //loop
  initialize
  REPEAT
    exit'now:=FALSE
    entry'rtn
    select'operation
    IF skip=FALSE THEN verify'ans
  UNTIL exit'now=TRUE
  evaluation
  RESTORE
UNTIL TRUE=FALSE //endloop
//
PROC instructions
  PRINT AT 6,1: "Letters a,b,c,d,e,f,g,h,i,j"
  PRINT "are randomly paired with the digits"
  PRINT "0,1,2,3,4,5,6,7,8,9"
  PRINT "Your task is to determine which"
  PRINT "number is paired with each letter."
  PRINT
```

more»

```
  PRINT "To help, you may ask for the sum,"
  PRINT "difference or product of any two let-"
  PRINT "ters and the program will supply the"
  PRINT "numerical answer in the form of its "
  PRINT "paired letter. Thus, for example, you"
  PRINT "may ask: c+f or a-d or b*g. If the"
  PRINT "solution is greater than two digits,"
  PRINT "the program will supply a letter"
  PRINT "which stands for the unit's digit."
  continue
  PRINT AT 6,1: "For subtraction, a negative"
  PRINT "answer will be given in its positive"
  PRINT "form. When you think you know the"
  PRINT "value of a letter, type it in this"
  PRINT "form:  c=6 ... The program will tell"
  PRINT "you if it is correct. The number of"
  PRINT "inquiries will be recorded and once"
  PRINT "you have identified them all"
  PRINT "correctly, you will be graded"
  PRINT "accordingly."
  PRINT "Pencil and paper can be helpful in"
  PRINT "solving the puzzle."
  continue
ENDPROC instructions
//
PROC heading
  PRINT AT 11,15: "MULTIDENT"
  PRINT "      A Game Whose Solution Involves"
  PRINT AT 13,12: "Number Properties"
  PRINT AT 19,22: "By Bill Inhelder"
  FOR i:=1 TO 2000 DO NULL
  PAGE
ENDPROC heading
//
PROC initialize
  count:=0
  FOR n:=0 TO 9 DO READ a$(n)
  FOR k:=0 TO 9 DO
    r:=RND(0,9)
    h$:=a$(k)
    a$(k):=a$(r)
    a$(r):=h$
  ENDFOR k
  FOR n:=0 TO 9 DO READ c$(n)
  FOR n:=0 TO 9 DO d(n):=0
  DATA "a","b","c","d","e","f","g","h","i","j"
  DATA "0","1","2","3","4","5","6","7","8","9"
ENDPROC initialize
//
PROC entry'rtn
  REPEAT
    errorfree:=TRUE
    count:+1
    INPUT "Enter arithmetic statement: ": b$
    IF b$(1:1)>"j" OR b$(3:3)>"j" THEN
      errorfree:=FALSE
      PRINT "Letters beyond J aren't allowed."
    ENDIF
    IF b$(2:2) IN "+,-,*,=" THEN
      NULL
    ELSE
      PRINT "You may only use +,-,*,="
      PRINT "No blanks are allowed between"
      PRINT "the letter and the operation."
      errorfree:=FALSE
    ENDIF
  UNTIL errorfree=TRUE
ENDPROC entry'rtn
//
PROC select'operation
  skip:=TRUE
  IF b$(2:2)="+" THEN add'rtn
  IF b$(2:2)="-" THEN subtract'rtn
  IF b$(2:2)="*" THEN mult'rtn
  IF b$(2:2)="=" THEN skip:=FALSE
ENDPROC select'operation
//
PROC add'rtn
  s:=0
  FOR m:=1 TO 3 STEP 2 DO
    FOR k:=0 TO 9 DO
      IF b$(m:m)=a$(k) THEN s:+k
    ENDFOR k
  ENDFOR m
  s:=s-10*INT(s/10)
  PRINT a$(s)
ENDPROC add'rtn
//
PROC subtract'rtn
  s:=0
  FOR m:=1 TO 3 STEP 2 DO
    FOR k:=0 TO 9 DO
```

more»

```
      IF b$(m:m)=a$(k) THEN s:=ABS(s-k)
    ENDFOR k
  ENDFOR m
  PRINT a$(s)
ENDPROC subtract'rtn
//
PROC mult'rtn
  s:=1
  FOR m:=1 TO 3 STEP 2 DO
    FOR k:=0 TO 9 DO
      IF b$(m:m)=a$(k) THEN s:=s*k
    ENDFOR k
  ENDFOR m
  s:=s-10*INT(s/10)
  PRINT a$(s)
ENDPROC mult'rtn
//
PROC verify'ans
  k:=0
  kk:=-1
  REPEAT
    IF a$(k)=b$(1:1) THEN kk:=k
    k:+1
  UNTIL k=10
  IF kk<0 THEN
    PRINT "Sorry, that is not correct."
  ELSE
    IF b$(3:3)=c$(kk) THEN
      d(kk):=1
      PRINT "Yes, that is correct."
      k:=0
      WHILE NOT (d(k)=0 OR k=10) DO
        k:+1
      ENDWHILE
    ELSE
      k:=0
      PRINT "Sorry, that is not correct."
    ENDIF
  ENDIF
  IF k=10 THEN exit'now:=TRUE
ENDPROC verify'ans
//
PROC evaluation
  PRINT "Congratulations! You got them all."
  PRINT "It took";count;"tries. Your rating is:"
  IF count<20 THEN PRINT "Excellent!"
  IF count>=20 AND count<30 THEN PRINT
  "Good"//wrap line
  IF count>=30 AND count<40 THEN PRINT
  "Fair"//wrap line
  IF count>=40 AND count<50 THEN PRINT
  "Poor - try again"//wrap line
  IF count>=50 THEN PRINT "Embarassing!"
  PRINT
  INPUT "Would you like to try again (y or
  n)?": ans$//wrap line
  IF ans$ IN "Nn" THEN STOP
  PRINT "You can improve your score by using"
  PRINT "the hint in the name of the game:"
  PRINT " MULTIDENT"
  continue
ENDPROC evaluation
//
PROC continue
  PRINT
  PRINT "    Press C to Continue"
  REPEAT
    cc$:=KEY$
  UNTIL cc$ IN "cC"
  PAGE
ENDPROC continue ■
```

```
USE c128 // HiRes Cube by Gary Parkin
setgraphic80 // use with Superchip
clear80 //, on C128 only
rec80(100,100,50,50)
rec80(150,125,50,50)
line80(100,100,150,125)
line80(275,75,225,50)
line80(100,50,150,75)
line80(225,100,275,125)
//
PROC rec80(x,y,width,height) CLOSED
  USE c128 // draws on unit "7:"
  width:=width+height+(height DIV 2)
  line80(x,y,x+width,y)
  line80(x+width,y,x+width,y-height)
  line80(x+width,y-height,x,y-height)
  line80(x,y-height,x,y)
ENDPROC rec80 ■
```

COMAL Commentary

by Alan Jones

COMAL is an excellent product that seems to be a classic case of poor marketing. I have to believe that if COMAL was an Abacus, Electronic Arts, or whatever product, that it would be a spectacular financial success. The COMAL Handbook [*now replaced by the Common COMAL Reference*] is an indispensable reference and it should be on the shelf of every Waldenbooks store. Do not give up! [*I tend to agree ... COMAL needs marketing by a big company. We would like to just support users, and let the other big company do the selling of COMAL itself, but thus far, no large company is willing to do this. Commodore, IBM, and Abacus each have turned down the chance to sell COMAL in the USA.*]

The price of the COMAL 2.0 cartridge is a major problem. If someone asked me about COMAL, I would recommend the CP/M version, assuming that they had a CP/M machine or were buying one anyway. What is needed is a lower cost, made in the USA, cartridge based version for the C64. You could put it on a single 27C512 EPROM, fix the shortcomings of the UniComal 2.0, and sell a cheaper and better implementation on both sides of the great pond. Or, you might consider getting a license to just copy UniComal 2.0 into cartridges here.

I am interested in UniComal COMAL for the IBM PC but $600 for the complete system is too much. [*We agree! We have repeatedly advised UniComal of this. However, the price is their decision, as we only import it.*] I can get a good FORTRAN compiler for a lot less!

I also do not think much of your current submission policy. A duplicate copy of COMAL Today and its disk is of no value to a subscriber. If the submitter is a current subscriber, you should extend the subscription by one issue. [*I agree! The policy is hereby changed, effective immediately!*]

Your report from Brian Grainger mentioned COMAL on the Archemedes. I looked into this a while back. The Archemedes has a 6502 emulation mode and the COMAL they use runs in this mode. It is not a significant development.

Grey Cartridge Conversion

I have successfully converted my 2.01 cartridge. Thanks for sending the conversion info. I talked with Terry Ricketts and learned a few more things. The overlays 5 and 6 can still be used after the modification. It can even be modified to use 27512 EPROMs. Of course the straight conversion will give an extra 64K of expansion, but it is nice to know that the conversion does not negate any of the other expansion options. The CMOS 27C256 EPROM is compatible with the existing circuitry and will draw less current. The instructions referred to a 74LS163 chip and my cartridge has a 74LS161. Terry said that the 163 is a binary counter and the 161 is a decimal counter. As used here it is not important and the modification instructions are valid for both chips.

The hardest part of the conversion is cutting two traces on the top side of the circuit board that are covered by the sockets. Instead of desoldering and removing two 28 pin sockets, I cut these traces by drilling holes through the board from the bottom side. The rest of the conversion was done according to the instructions.

I used the CMOS 27C256 EPROMS. In spite of using a more static sensitive device, working in the dry December air, not using any special antistatic devices, and occasionally touching the pins with my fingers, I got the chips programmed and installed on the cartridge without any damage. While all due care should be taken, anyone contemplating the cartridge conversion should not be put off by reading the usual cautions. The 16K Superchip would not

more»

work on the modified cartridge and had to be copied into the lower half of another 27C256 EPROM. The modified cartridge with EPROMS copied from the grey cartridge will not pass the black cartridge test, but still passes the grey test. The Rabbit on Superchip seems to work fine on the modified cartridge. Is there a program to test Superchip? [*Yes, the Superchip test programs are on the Superchip On Disk disk and the Superchip Programs disk*]

These notes may be helpful for anyone else contemplating modification of the 2.01 cartridge.

Double Precision Math

Aw #$X&! It seems that I sent in an early development version of the double precision math routines by mistake. As published in *COMAL Today #20* the routines will probably work well enough for the timid, and have probably already been fixed by true programmers. In the four main procedures there is a test for small values for underflow protection:

IF ABS( )>0 THEN, or ABS( )<0

0 is really too small and I eventually decided that 1 was a reasonably small value. In **longmul**, **scal2** should have been **scale2**. In **longadd2**, **xx:=(((x-z)**... should be **xx:=(((z-x)**... This last one is the only potential problem but it is protected by the **ABS(z)>0** error. The final version that I intended to send in is included in the iterative improvement program (on *Today Disk #22*). I have also included a double precision square root procedure, **dsqr**.

Blessed Be The BLAS

The **BLAS** stands for Basic Linear Algebra Subroutines. They are a set of versatile subroutines that are often used in software for solving linear systems, and other matrix problems. But this is not about the BLAS.

Nick Higham wrote a BLAS package and shared it with the rest of the world. I found it on the European Disk Set along with some excellent documentation, and an example linear system solver program. The BLAS is also in Packages Library Vol. 2, although the documentation is shortened. He also gave his address and requested that anyone who finds the routines useful, or has a potential improvement send him a note.

It had been more than 18 months since he wrote the package, and I planned to use the BLAS quite a bit, so I wrote him a letter to see if he had any updates. I also wanted to understand the "*different rounding*" problem between assembled code and the equivalent COMAL code. He sent back a reply and informed me that the BLAS version that I had is still the latest and greatest version. And, he reassured me that the "*different rounding*" problem is in favor of the BLAS. The accumulator in the 6502 (or 6510) processor actually has 9 bits counting the overflow or carry bit. And sometimes it behaves as a guard digit. When accumulating a floating point sum we essentially have a 33 bit mantissa which is rounded back to 32 bits when the value is stored. The BLAS and other packages that use the kernal routines always use proper rounding, never chopping, and when they differ from the equivalent COMAL code they can be considered to be more accurate.

I also mentioned that I had written some double precision procedures and was planning to submit a program using his BLAS to show the iterative improvement technique. He then paid an extra dividend and referred me to a relevant paper by Skeel on the merits of iterative improvement with single precision. But this is not about Skeel.

Nick also mentioned a very surprising fact. Even after more than 1½ years, I was the first

more»

person to write to him about his BLAS. The BLAS are very useful, especially when assembled into a package. Is it possible that Nick and I are the only two C64/COMAL 2.0 users on two continents that can appreciate the BLAS? I think too many computer programmer hobbyists are simply accepting free software as their right for attending a users group meeting or buying a disk from a users group. I think they also have an obligation to say thank you, especially if the author was thoughtful enough to give his address and an invitation to write.

Most of us write programs for our own personal use. Then we realize that there is no similar program generally available and other people may find it useful. So we clean the program up, make it a little easier to use, add additional comments that we don't need, and write some documentation, an accompanying article, etc. The problem is this usually takes longer than developing the program did. I will bet that if an author does not get any feedback about his software that he will conclude that he is wrong in his assessment of what is useful to others, and refrain from doing extra work to share his programs with others. On the other hand, a little response might encourage more good software, and other dividends.

Nick is not using his C64 anymore, but I don't think users of more expensive computers are any more responsive. Have you ever written to a software author whose program you used?

Iterative Improvement

Iterative improvement is a technique that can be used to improve the solution of a linear system of equations. A program on *Today Disk #22* is presented to solve linear equations by this technique. It is also set up to explore roundoff error by solving Hilbert matrices. It uses the BLAS package and double precision math procedures.

Consider the problem of solving A(,)*x()=b() for x(). The usual solution is to use Gauss elimination with partial pivoting. In modern terminology this the same as using the LU factorization of A and using backsubstitution to solve the problem. L is a lower triangular matrix, and U is an upper triangular matrix. Pivoting is necessary to stabilize the solution against roundoff errors and the pivoting is accounted for by a permutation matrix P. We then factor PA = LU. This operation requires 1/3 n 3 flops or multiplications. Actually P is never formed and is represented by an integer vector. In general, we ignore the P and just assume that it is used. We then solve LUx=Pb in a two step process. Solve Ly=Pb for y, then solve Ux=y for x. Pb is just a reordering of the vector b. L is lower triangular so y can be solved directly by forward substitution. U is upper triangular so x can be solved directly by back substitution. The solution following the LU factorization requires only n 2 flops. Most of the computation time is taken in the LU factorization and we will take advantage of this fact.

Unfortunately, after all of that work x may not be as accurate as we would like. The accuracy is limited by the precision of the arithmetic and the condition of the problem. I do not want to give a detailed discussion of norms and condition numbers here. However, the condition number of a matrix is an indication of how difficult the problem is. A condition number of 1.0 is easily solved and a large number means that the solution will not be accurate to full precision. When the condition number K multiplied by the numerical precision (1/2 32) is greater than 1.0 the problem may be numerically singular. Grinding through the algorithm will yield unreliable results or fail entirely. A truly singular matrix has no unique solution. In effect the condition number of the matrix gives an indication of how many digits of accuracy will be lost in the solution. To improve the accuracy

more»

of the solution we must increase the math precision or decrease the condition number.

Decreasing the condition number is not usually an option since this means changing the original problem. One way to do this is by scaling the problem. This is rarely done except on an ad hoc basis. What we want to do is in some sense make all of the elements of the matrix have similar magnitudes. This is essentially done by choosing appropriate units for the variables in the x and b vectors. That is, pick units of Km/sec., microvolts, or whatever so that the matrix looks nice.

If single precision math does not provide an accurate enough solution, the obvious choice is to use double precision. Double precision routines are very slow unless implemented in hardware. In the case of the COMAL procedures they are extremely slow. We certainly do not want to solve an 1/3 n 3 + n 2 problem in double precision if we do not have to. Let us assume that we have a problem whose condition will cause a loss of 40 bits of precision and we need a solution with at least 16 bits of precision. Normal 32 bit real math will not work at all, and double precision math will be required for the entire solution giving us 64-40=24 bits of accuracy. Suppose instead that we have a problem that only causes a loss of 24 bits of accuracy. With normal precision math we would expect 32-24=8 bits of accuracy in the solution. This is a problem that can be readily solved with the iterative improvement technique.

The technique works as follows. First copy the matrix A into a working matrix AA. AA will be decomposed into its LU factorization with normal precision math. Solve for the solution vector, xest in the usual manner. At this point xest should have about 8 bits of accuracy. Now check the solution by calculating the error residual vector r:=A*xest-b. Then solve A*delx=r for delx. Now improve xest with xest=xest-delx. In effect the error has been fed back into the system and corrected. The improvement process is repeated until the result is accurate enough, or until it has converged to full 32 bit precision.

In order to make this work, the original A must be used along with double precision in calculating the residual r. r is then rounded to normal precision for the next step. Note also that we already have the LU factorization of A in AA and solving for delx just takes n 2 normal precision flops. On each iteration we then have n 2 calls of exactmul2, a double precision product of two normal precision numbers, and n*(n+1) calls of longadd2, for the double precision add, and n 2 single precision flops. This is much faster than performing the LU factorization in double precision.

In general, if the initial solution has d bits of accuracy, then each iteration will be expected to add about d more bits of accuracy. This is called a linear convergence rate. For the last hypothetical problem, one iteration might give the needed 16 bits of accuracy and a second iteration could assure it. Five iterations would probably give full 32 bit accuracy in the solution. If the problem was such that the initial solution only had 2 bits of accuracy we would expect convergence to 32 bits after about 16 iterations. In this case it might be better to solve the whole problem in double precision.

The iterative improvement technique expands the class of linear systems that we can solve. The technique can also be applied to other problems.

Iterative Program

The iterative program on *Today Disk #22* was written to test all of the procedures that it uses and also to be a tool for exploring roundoff error and the solution of illconditioned matrices. It uses the BLAS package and solution routines written by Nick Higham. sgiem is the

more»

iterative improvement procedure. All of these routines are based on the standard LINPACK FORTRAN subroutines. An updated version of the COMAL double precision math procedures are also included.

A Hilbert matrix is formed and then solved. The Hilbert matrix has elements A(i,j):=1/(i+j-1). Where i and j go up to n, the order of the matrix. The ith element of vector b is the sum of the ith row of A. In the absence of roundoff error, the exact solution of Ax=b is 1 for all elements of x. The Hilbert matrix is easy to form and its condition number increases rapidly with n. Examination of the numerical solution of x gives an immediate indication of the precision of the arithmetic used. When the Hilbert matrix is setup as defined above a slightly different problem is actually formed. For example 1/3, 1/5, etc. can not be represented exactly on a binary arithmetic machine. Thus the exact solution of the problem actually formed will not be exactly 1. We are attempting to calculate a very good approximation of the exact solution, not the problem we only thought we were solving. In order to start with a clean slate we can form A(i,j):=one/(i+j-1), where one is a large number that allows the problem to problem to be represented exactly with binary numbers. A value is given in the program that allows exact representation of the problem up to n=9. And the iterative improvement algorithm will converge to the exact solution of 1 to full normal precision.

The procedure sgeim is fairly close to the sgeim subroutine given in LINPACK. It will iterate to full convergence or terminate after a fixed number of iterations. If it terminates on the iteration count the result is unreliable and info returns the value -1. If it converged, info returns the number of iterations taken. For a specific application you may want stop iteration after convergence to a specified accuracy or even after only one iteration. There are also a few lines of code to calculate and print some extra stuff for your amusement. You might want to set dpr#:=FALSE to use only single precision math in calculating the residual.

Skeel wrote an interesting paper on the merits of using iterative improvement with a normal precision residual calculation. He performed a numerical analysis of the problem to attribute errors to scaling and condition separately and concluded that using single precision would eliminate the error that was do to bad scaling. Thus, instead of trying to scale a problem prior to solution you could just take a step of iterative improvement. He also suggested that it could be used in place of the partial pivoting in the LU factorization. I think this last idea is stupid. I have not experimentally verified the other claim, but this program is a good tool to try it on.

The program will prompt you for a value of n. It will calculate and display the result from sgeim and if successful it will use the result to calculate the error in the noniterative solution. Try different values of n, one, and dpr#. Note the results, number of iterations, estimated condition number, and error. You can also try different matrix problems or try to badly scale the Hilbert matrix to see what effect that has.

You should LIST the separate procedures to disk so that you can use them in your own applications (via the MERGE command). I should also mention that Gauss Elimination or LU factorization is not the only way to solve linear systems. There are other methods that are more accurate but slower, or faster for special problems. For solving illconditioned systems the singular value decomposition is unsurpassed and provides more information about the problem. For more information you should consult a book on numerical analysis or the LINPACK Users' Guide. - Alan Jones, Ames, IA ■

GOTO

by Peter Burkinshaw, TeleNova
Box 213, S-14901, Nynashamn, Sweden

Many thanks for the *COMAL Todays* that we have received. Reading them makes it hard to believe COMAL was a European *invention*! We still have a lot to learn from the US about how to raise the visibility of what we're doing, and communication in general.

After reading *COMAL Today #6*, I felt that some kind of counter arguments were needed to the wave of anti-GOTOism that seems to be sweeping the world, so I have taken the liberty of sending you my views as a programmer of some 30 years experience on about 10 different hardware models and a dozen high level languages.

[*Yes, this article is from the summer of 1985! It has a unique history of disappearing from our files and disks. Originally we were planning some comments on each point made in the article. It was worked on for 3 months, then was misplaced. Before we could redo our efforts, the original was lost as well. A few months later, it was out of our plans. I still don't know where the original or our edited version is, but I just found an old printout from the original disk file. I cannot find that disk, so I am retyping the whole thing. Views in this article are not necessarily those of COMAL Users Group USA Ltd. Our no-GOTO policy has been maintained during the nearly 3 years of time after this article was originally written. Now, here is the rest of the story...*]

It is a sad reflection on our ability to think for ourselves, that the opinions of some of our leading academics should become gospel without the burden of proof being felt to be necessary.

It is certainly true that programs can be written in hard to understand ways, one of which is the use of the unnecessary GOTO statements. At this point of my argument, I am usually assailed by Red Guards chanting the thoughts of you know who, who proved that any program can be written without the use of a single GOTO. (I can also travel from Boston to San Francisco on a bicycle, but that in itself does not make it essential, or even desirable.)

What we should consider objectively and carefully is: *When is GOTO a useful statement?* I have examined many programs, in a large range of applications and languages, to find an answer to this question. I have prepared a list below of the circumstances in which I find the GOTO statement too valuable to pass up.

Historically, the use of GOTO has been forced in many cases by the structure of the programming language. In COMAL, for example, more powerful **IF** and **CASE** statements obviate the need for writing the sort of

IF «*condition*» THEN GOTO xxx

that many BASICs oblige you to write. The use of labels, incidentally, had been established in the earliest assemblers, (such as IBM 650 S.O.A.P. and Ferranti Pegasus Autocode, back in 1957), because of their higher degree of memorability and information redundancy. So how come many of the so-called higher level languages like Fortran, BASIC, and Pascal are still bumbling along with numbers that are so easily misread, mistyped and generally screwed up? Thank God for the appearance of that much sneered at language COBOL, around 1961. It has lots of snags of course, but it is a pity that the possibility of writing proper labels was overlooked by many later language designers.

Getting down to brass tacks, what are these necessary GOTOs that I won't give up writing?

The first, and most obvious use, is to achieve code commonality between different parts of a compound IF structure, with a minimum of overhead, both on me and the hardware. Any "*commercial*" programmer will recognize the sort of decision-table situation in which 2 to the

more»

umpteen possible combinations reduce to about 20 different actions, some of which include one or more of the others as subsets.

Where the terminal actions are common, the most effective way of handling them is to write a GOTO in one part of the program to "*join*" it to the other. I have heard all the rival structured alternatives, like writing endlessly complicated IF statements, to determine whether or not to perform the code in a simple one level sort of way like:

```
IF black AND NOT round AND thin AND metal
  AND weight<5 OR white AND square AND
  wood AND ... AND ... OR grey AND flat AND
  NOT heavy ...

THEN DO something

ELIF black AND NOT round AND thick AND
  metal AND ... OR ...
```

This is surely the way to madness and unmaintainability.

The objections are fairly obvious. The programmer uses four pencils a day and suffers permanent numbness in the fingers, the computer spends ten times more evaluating and stacking and re-evaluating boolean expressions than anything else. What I write is something like:

```
IF black THEN
  IF round THEN
    IF thin THEN
LABEL1:
      do something
    ELSE
LABEL2:
      do something else
    ENDIF
  ELIF square THEN
    IF thin THEN
      GOTO label2
    ELSE
      GOTO label1
    ENDIF
  ELSE
    etc...
```

I can do this in COMPIS COMAL which does not presume to teach me how to write programs by preventing me writing this kind of GOTO.

The next suggestion I get is why not duplicate the **do something** in each place they occur? The answer is, because I don't like writing the same thing twice (or more). It's not just my natural laziness, it's an offense against the laws of good software engineering --- never define things twice. You'll get one wrong, and even if you don't you'll still have to maintain all the repetitions, and one day somebody (not you, of course) will forget to change just one of them, because it was over the next page of the listing and got overlooked. So another million dollar machine falls into the sea, because it was **rainy AND windy AND NOT cold AND height>30000 AND course$="NNW" AND speed<200 AND NOT red'alert...**

The final and most seductive alternative is a variation of the second. Namely, that I make the requisite three lines of code a subroutine (procedure, to those born after 1957) and call it in all the places I want to use it. This might sound fine, but leads to a disease called procedural fragmentation in which everything is beautifully tree-structured. The problem is you can't see what the heck is going on, because it's all nested procedure declarations, parameter lists and import lists and no action!

It's the same in the program at run time. What's mainly happening is the frantic pushing and popping of procedure parameters and return addresses. About once in a blue moon, it gets to add a couple of numbers together, then off into the stack-madhouse again.

more»

The ultimate refuge of the advocates of GOTO-less programming is to proclaim my program unreadable, because they weren't trained to read such programs. Ignorance being elevated to a virtue by those who seek to instruct others!

The second and simpler to understand use of GOTO is to escape from a FOR loop when something unusual happens. I get a variety of interesting alternatives to this one, like:

"*Construct a WHILE or REPEAT loop instead.*"

A FOR loop runs faster, and I prefer not to have to manage the loop control variable myself, thanks!

"*use EXIT*"

Interesting. All the languages that have implemented EXIT take you to the statement after the NEXT (or next NEXT in ADA). Of all the places I want to be in my program, about the least likely is where I would have been if the loop had run to a "*successful*" conclusion. Because I then have to figure out how I got there. An action not exactly helped by languages (ADA, Commodore COMAL 2.0) that de-scope the loop control variable (the x in FOR x=...) as soon as the loop ends, so you can't even test the final value without doing an assignment every time around, or duplicating the end-of-loop test.

The third example is a variation of the second. I have two nested FOR loops, and if a certain condition arises in the inner, I wish to restart the outer with its current control variable value:

```
FOR x=1 TO lim DO
rerun:
  ...
  FOR y=x+1 to lim2 DO
    ...
    IF «cond» THEN GOTO rerun
    ...
  ENDFOR y
  ...
ENDFOR x
```

No amount of EXITing helps you here. You can only avoid the GOTO by totally abandoning the nested FOR loops and writing a rather messy set of nested IFs and REPEAT ... UNTILs, with lots of duplication. What you need, if you prefer, is a new sort of statement like "AGAIN x".

The fourth example is a more complex form of the third. This time I have two loops, a REPEAT ... UNTIL in the middle of which a FOR loop starts, whose NEXT is after the UNTIL:

```
REPEAT
  READ y
  ...
  FOR x=1 TO y
  ...
  IF «cond» THEN
    UNTIL eod
  ELSE
    ...
  UNTIL p<q
NEXT x
```

No language I know allows you to weave loops like that, so you make the REPEAT a label, and the UNTIL another IF ... THEN GOTO. The GOTO-less brigade suggest I destroy what structure exists here and replace it by a less efficient, and more time consuming set of nested IFs for the only reason that they think it looks better.

My final example concerns a REPEAT loop with two alternate UNTILs:

```
REPEAT
  READ a
  ...
```

more»

```
IF «cond» THEN
  ...
  READ b
  ...
  UNTIL f(a)>f(b)
  ...
ELSE
  RESTORE somewhere
  READ c
  ...
  UNTIL g(a)=g(c)
ENDIF
```

I cannot create a simple boolean OR condition in the UNTIL, because I can't simultaneously evaluate both conditions. I may only make one or other test, or I get a no-value error or worse, wrong results, in a less helpful language. This one usually leaves them speechless!

The problem arises because we are trying to represent a dynamic process using static scoping concepts (which is one reason, incidently, why dynamic store management cannot be married to static scoping rules without creating impossible restrictions on the user.) Who said there could only be one UNTIL or one NEXT in a loop?

We allow more than one RETURN in a PROC or FUNC for precisely the same reason: there is more than one dynamic end-point in the process, and static structure freaks will tie themselves in knots to avoid admitting their pet language is, like virtually all besides, defective for the very purpose for which it was ordained.

To those who may suspect that I made up all these examples, I would simply say that they are all taken from real programs written by myself for real companies. I have only paraphrased them into COMAL for ease of understanding.

I do not accept the right of anyone, from professors down, to make ex-cathedra statements about what is good and what is bad without documented, statistically valid measurement. Or expect anyone, including their captive student audiences, to take them as established fact. We do, after all, talk about computer science and software engineering, not computer faith and software superstition.

Further Reference:

From COMAL Today issues:

Programming Languages For Beginners, Borge Christensen, #1, page 8
The COMAL Column - The Reply, Mindy Skelton #3, page 20
Where Have the GOTOs Gone, Raymond Quiring, #6, page 64
Goodbye GOTO, Valerie Kramer, #6, page 78
On GOTO, #11, page 72
Keywords, #11, page 33
Scope Rules, Richard Bain, #14, page 60
Scope Rules, #17, page 10
COMAL Kernal, #17, page 48
A Matter Of Style, Bill Inhelder, #18, page 17
A Good Case, B K Ball, #18, page 8
A Matter Of Style, Alan Jones, #19 page 8

From Books:

Structured Programming With COMAL, by Roy Atherton, pages 98, 139, 144
Beginning COMAL, Borge Christensen, page 92
COMAL Handbook, Len Lindsay
1st Edition, page 84
2nd Edition, page 149
Acornsoft COMAL manual, page 307
COMAL From A to Z, Borge Christensen, pg 31
Starting With COMAL, Ingvar Gratte, page 107
Common COMAL Reference, Len Lindsay, pg 110
Foundations With COMAL, John Kelly, page 334
CP/M COMAL manual, page R-31
IBM PC COMAL manual, page 160
IBM PC COMAL tutorial, -not mentioned-
Introduction to Computer Programming With COMAL, J William Leary, -not mentioned-
Mytech COMAL manual, page 21 ■

Stats For Teachers

by Gerard Frey

Most of my statistics programs were first written in BASIC. Then I heard about COMAL and decided to try to re-program them. What a pleasure it was! In fact it was downright fun! I am sending the Chi-Square as an example to you. These programs will help teachers of small classes (less than 100 students) to do basic statistical analyses on the test results of their students. It may also assist students in their smaller research projects. This limitation in size is due only to the small RAM available for data processing. This program will read in data arrays and calculate the Chi-Square and the Degrees of Freedom.

There are several options offered for each statistical analysis. Normally the output is sent to the screen where errors can be edited before a printout is made. A quit option is provided.

In addition to this program I have several other statistical programs which make data analysis for students and teachers a thrill. Term marks and final results are quickly processed. Using COMAL also resulted in neat and clear printouts of data and tables. The following are now ready for use:

Means and Medians(rows/columns)
Standard Deviation(rows/columns)
Sorting(rows/columns)
Correlation-Pearson(columns)
Correlation-Frey(columns)
One-Way ANOVA(t-test)
Two-Way ANOVA
Analyzing Rating Scales
Chi-Square test
Sample size

If there is enough interest in these programs, I hope to continue with others that may also be helpful in handling data. At the moment only complete and balanced arrays are handled.

Data are prepared in DATA statements and listed to disk with the LIST command and then MERGED with the appropriate Statistcal Program for analysis purposes. For some data, several kinds of analyses are possible. In some cases it may be better to SAVE the data and LIST the programs.

Anyone interested in receiving a disk of these programs should send a postal note for $20.00 U.S. or $25.00 Canadian to: Gerard Frey, 38 Corkstown Road, Nepean, Ont. Canada K2H 5B4

CHI-SQUARE Program

```
// COPYRIGHT 1987 - GERARD FREY
// Ottawa, Canada - December 1987
// Permission to publish in COMAL Today
// granted in Dec 16, 1987 letter.
PROC give'inst
  PAGE
  PRINT "   THIS PROGRAM CALCULATES"
  PRINT "      THE CHI-SQUARE"
  PRINT "   **********************"
  PRINT " Select from OPTION TABLE."
  PRINT "   You SAVE the DATA in"
  PRINT "  ROWS of DATA STATEMENTS"
  PRINT "        such as:"
  PRINT
  PRINT " 1000 DATA ""Resp#"",12,25,34"
  PRINT
  PRINT " for observed frequencies."
  PRINT "    THEN you MERGE the"
  PRINT "    PROGRAM to the DATA."
  PRINT "   Use the MERGE command."
  PRINT "    Enter the number of"
  PRINT "  ROWS(resp) and COLUMNS"
  PRINT "  (categ) at the prompt."
  PRINT "    HAPPY CALCULATIONS!"
  wait'for'key
ENDPROC give'inst
//
PROC set'up
  PAGE
  PRINT "         BASIC STATISTICS"
  PRINT
```

more»

```
  PRINT AT 3,4: "INPUT Data Information"
  INPUT AT 5,4: " ENTER the SUBJECT? ": sb$
  INPUT AT 6,4: " ENTER the GROUP?   ": gp$
  INPUT AT 7,4: " ENTER the DATE?    ": dt$
  INPUT AT 9,4: " How many rows?     ": rows
  INPUT AT 10,4: " How many columns? ": cols
  // Data Dimensions //
  DIM item$(rows) OF 10
  DIM freqob(rows,cols), freqex(rows,cols)
  DIM sumrows(rows), sumcols(cols)
ENDPROC set'up
//
PROC read'data
  PRINT
  no#:=0
  FOR x:=1 TO rows DO
    no#:+1
    READ item$(no#)
    FOR y:=1 TO cols DO
      READ freqob(x,y)
    ENDFOR y
  ENDFOR x
ENDPROC read'data
//
PROC print'data
  PAGE
  ZONE 2
  PRINT AT 2,8: "  ::CHI-SQUARE ::"
  PRINT AT 3,8: "  :: OBSERVED  ::"
  PRINT AT 4,8: "  ::FREQUENCIES::"
  PRINT AT 6,4: gp$,sb$,dt$
  PRINT
  print'nos(item$(),rows,cols,freqob(,))
  PRINT
  wait'for'key
  ZONE 0
ENDPROC print'data
//
//*** MAIN PROGRAM ***//
work'color
hard:=0
give'inst
set'up
LOOP
  PAGE
  SELECT OUTPUT "ds:"
  show'menu
  wait'for'key
  CASE reply$ OF
  WHEN "1"
    end'work
  WHEN "2"
    RESTORE
    read'data
    print'data
  WHEN "3"
    RESTORE
    read'data
    chi'square
  WHEN "4"
    RESTORE
    read'data
    SELECT OUTPUT "lp:"
    print'data
    SELECT OUTPUT "ds:"
  WHEN "5"
    RESTORE
    read'data
    SELECT OUTPUT "lp:"
    chi'square
    SELECT OUTPUT "ds:"
  OTHERWISE
    PRINT "Illegal reply.."
    wait'for'key
  ENDCASE
ENDLOOP
//
PROC show'menu
  PAGE
  PRINT
  PRINT "       **** OPTION TABLE ****"
  PRINT "       *     CHI-SQUARE     *"
  PRINT "       *                    *"
  PRINT "       * <1> QUIT           *"
  PRINT "       * <2> LIST THE DATA  *"
  PRINT "       * <3> SHOW CHI-SQUARE *"
  PRINT "       * <4> PRINTOUT DATA  *"
  PRINT "       * <5> PRINTOUT CHI-SQ *"
  PRINT "       *                    *"
  PRINT "       **********************"
ENDPROC show'menu
//
```

more»

```
PROC end'work
  PAGE
  END "    ::TO REPEAT - PRESS <F7>::"
ENDPROC end'work
//
PROC wait'for'key
  SELECT OUTPUT "ds:"
  PRINT
  PRINT "      ***********"
  PRINT "      PRESS A KEY"
  PRINT "      ***********"
  REPEAT
    reply$:=KEY$
  UNTIL reply$<>CHR$(0)
  PAGE
  PRINT
ENDPROC wait'for'key
//
PROC chi'square
  PAGE
  ZONE 2
  PRINT
  totrows:=0; totcols:=0
  FOR x:=1 TO rows DO sumrows(x):=0
  FOR y:=1 TO cols DO sumcols(y):=0
  ro#:=0
  FOR x:=1 TO rows DO
    ro#:+1
    FOR y:=1 TO cols DO
      sumrows(x):=sumrows(x)+freqob(x,y)
    ENDFOR y
    totrows:=totrows+sumrows(x)
  ENDFOR x
  co#:=0
  FOR y:=1 TO cols DO
    co#:+1
    FOR x:=1 TO rows DO
      sumcols(y):=sumcols(y)+freqob(x,y)
    ENDFOR x
    totcols:=totcols+sumcols(y)
    PRINT
  ENDFOR y
  PRINT AT 2,8: "::EXPECTED FREQUENCIES::"
  PRINT AT 4,4: gp$,sb$,dt$
  PRINT
  //Calculate Expected Frequency//
  ro#:=0
  co#:=0
  chi:=0
  PRINT "Categories",
  FOR y:=1 TO cols DO
    co#:+1
    PRINT USING "###": co#,
  ENDFOR y
  PRINT " Total"
  PRINT
  PRINT "          ",
  FOR y:=1 TO cols DO
    PRINT "---",
  ENDFOR y
  PRINT "------",
  PRINT
  FOR x:=1 TO rows DO
    ro#:+1
    PRINT item$(ro#),
    FOR y:=1 TO cols DO
      freqex(x,y):=0
      freqex(x,y):=sumrows(x)*sumcols(y)/totrows
      PRINT USING "###": freqex(x,y),
    ENDFOR y
    PRINT USING " ####": sumrows(x)
    PRINT
  ENDFOR x
  PRINT "          ",
  FOR y:=1 TO cols DO
    PRINT "---",
  ENDFOR y
  PRINT "------"
  PRINT "Total     ",
  FOR y:=1 TO cols DO
    PRINT USING "###": sumcols(y),
  ENDFOR y
  PRINT USING "#####": totcols
  PRINT
  PRINT
  FOR x:=1 TO rows DO
    FOR y:=1 TO cols DO
      chi:=chi+((freqob(x,y)-freqex(x,y)))^2/
      freqex(x,y)//wrap line
    ENDFOR y
  ENDFOR x
  PRINT "    The CHI-SQUARE IS ",
```

more»

```
  PRINT USING "####": chi
  PRINT "    DEGREES OF FREEDOM ",
  PRINT USING "####": (rows-1)*(cols-1)
  PRINT " Consult Chi-Square for significance"
  wait'for'key
ENDPROC chi'square
//
PROC print'nos(ro$( ),totro,totco,number(,))
CLOSED//wrap line
  PRINT "Categories ",
  FOR cols:=1 TO totco DO
    PRINT USING "###": cols,
  ENDFOR cols
  PRINT
  PRINT "          ",
  FOR y:=1 TO totco DO
    PRINT "---",
  ENDFOR y
  PRINT
  FOR rows:=1 TO totro DO
    PRINT ro$(rows),
    FOR cols:=1 TO totco DO
      PRINT USING "###": number(rows,cols),
    ENDFOR cols
    PRINT
    PRINT
  ENDFOR rows
  PRINT "          ",
  FOR y:=1 TO totco DO
    PRINT "---",
  ENDFOR y
ENDPROC print'nos
//
PROC work'color
  USE system
  textcolors(6,11,0)
ENDPROC work'color
// CHI-SQUARE --- DATA FORMAT //
// MERGE Programs to DATA
// In Example below: Enter 3 for
// rows Prompt and 4 for columns
DATA "Response 1",20,5,15,20
DATA "Response 2",10,40,30,23
DATA "Response 3",20,20,10,25 ■
```

Football Quiz

Graphics by JR Mayhew
COMAL conversion by Luther & Dawn Hux

Football widows beware, here's another football game that will keep the NFL fans glued to the tube.

One method to get students really involved in programming is to have them write a quiz game of their own design. The subject matter varies greatly but many tend to choose sports. The teacher instructs them to plan the theme, write the questions and instructions, and decide what artwork will be used. Then the class goes through the lessons of writing a quiz. They learn about counters, testing the answers, writing data lines, presenting instruction screens and reward screens. Quiz programs are simple in structure and most students do a good job. But some programs really stand out. This quiz by JR Mayhew, a senior at Bethlehem Christian Academy, is an example of great keyboard graphics. The NFL football quiz concept is informative, instructional and is of wide general interest. The trophy, with sparkling highlights, is a real treat for the knowledgeable NFL fan who knows the correct team name for at least 26 of the 28 NFL cities presented. The team colors are used in the background for added graphic effect and to help you choose the correct team for cities with two teams.

At the time this program was written by JR we were still teaching BASIC. To capture the graphics for this COMAL program, we used the BASIC to COMAL converter by Sol Katz in *COMAL Today #13*. We then stripped the spaghetti logic and made each of the graphic screens a PROCedure. A COMAL quiz program was then written that called each graphic procedure as needed.

Due to the extensive use of keyboard graphics, this program is not listed here but is on *Today Disk #22*. Enjoy! ■

1581 Disk Drive

by Gary T. Parkin
200 La Cascata, Clementon, NJ 08021

I have been reading and using COMAL now since *COMAL Today #5*. I like the freedom that COMAL gives me and I am now teaching myself PASCAL. I could never have done it, if it were not for COMAL. I am experimenting with a CAD program in COMAL, and hope to improve it with the use of the 1351 Mouse. I will check out the mouse package on *Today Disk #18* and send in the CAD program when it is finished.

Commodore 1581 Drive

What is a 1581? Well, the 1581 is probably the best thing that Commodore has dreamed up so far. It's a disk drive that does almost everything a hard disk does (just not as swiftly) except cost you an arm and a leg - (it retails for about $235.00 [*$189 discounted*]). It's a 3 1/2" disk format that, unformatted can hold 1 Meg., and when formatted, can hold 808K of storage, about as much as 5 1541, and 2 1571 disks. [*One 1581 disk easily holds the entire HazMat shareware system written by Mark Finley! And the added speed makes it a fantastic system*] The compact drive size (8 1/2" X 2 1/2" X 5") is great for small spaces, and Commodore put a power supply outside of the unit just like that of the C128, so there won't be overheating problems.

The small disks are a bit more expensive but I find the added space and ease of use an advantage well worth the cost. It is compatible with the 1541 and the 1571, making it a good second drive for the C128 and C64. It is about 50% - 60% faster than the 1571. It has 2 dip switches in the rear to set the device as 8, 9, 10 or 11. Also, you can have up to 296 directory entries, (the 1541 & 1571 only hold 144 each). There is one more good feature - Disk Partitioning! Which means you can have sub directories of files. The main directory, we'll call it the "root" directory, can have it's own directories of files that behave like independent disks.

The sub directories have a file type of CBM, which means you can't load or run them like a file or program. The manual speaks of complete compatibility with CP/M software but at the time of this writing, I am still waiting for my new CP/M disks to arrive (your old CP/M cannot "see" the new drive).

I will now give you some examples of using the 1581 with COMAL. **These examples assume your 1581 is set to device #9.**

A Partition:
- Must start on sector 0
- Must be a multiple of 40 blocks (or 1 track)
- Must be at least 120 blocks long (or 3 tracks)

After formatting, if the Partition is to be a directory, the Partition will be 80 blocks in length (40 for the Partitions' BAM or directory)

On a multiple "Section" you must allow for a master directory.

Suppose we want to make a partition to hold:
1) CAD Program
2) CAD drawings

Suppose we want the CAD section to be 280 blocks long.

```
120 blocks CAD Program
120 blocks CAD Drawings
240 blocks
 40 blocks Partition Directory
280 blocks Total
```

So: CAD Program = 80 blocks Free
CAD Graphics = 80 blocks Free

By the way, a Partition name can be different from it's CBM file name, but we will make them the same. Let's start...

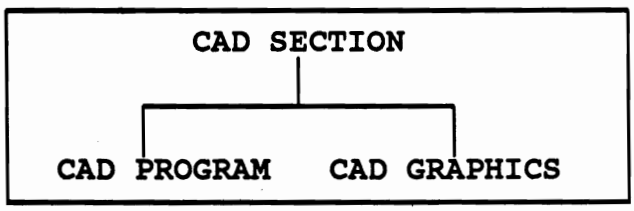
more»

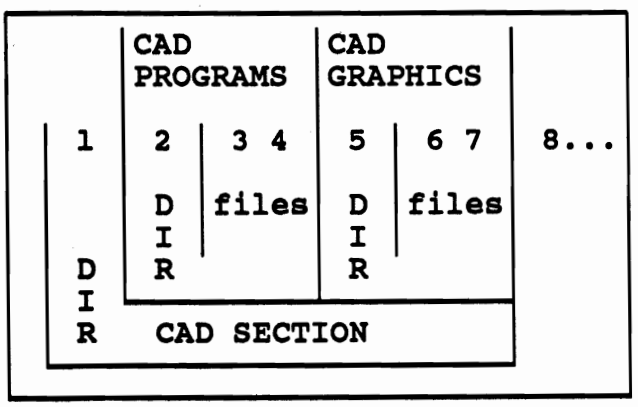
Rather than go into the inner workings of this machine, I have written a "*Partition Aid*" to help you (on *Today Disk #22*). Suppose we want a Partition file starting on track 1 named "CAD SECTION". Suppose we want a Second Sub Partition file named "CAD PROGRAM" (You must start on track 2 to protect the former directory). Suppose we want a Third Sub Partition file named "CAD GRAPHICS" (You must start on track 5 to protect the 2 former directories)

To get to any Partition, just type:
PASS "/:partition filename",9

To get to the "root" or main directory, type:
PASS "/"

Below is a drawing followed by a block count:





Any disk command that works on the 1541 and 1571 drives will work on the 1581. Using the default settings, both could be accessed as shown in the chart below:

1541/1571 (unit 8)	1581 (unit 9)
dir	**dir "2:"**
save "filename"	**save "2:filename"**
load "filename"	**load "2:filename"**
verify "filename"	**verify "2:filename"**
delete "filename"	**delete "2:filename"**

And for Superchip fans: **type("2:filename")**

Or, if you will be issuing several commands to your unit 9 1581 drive, you may change COMAL's default drive to be that drive:

UNIT "2:"

The only flaw I find with the 1581 is that the DOS SHELL program does not work well with the 1581.

Timetest

The results of my time tests shown below will give you a little bit more feel for the way the 1581 works (a different file was used for BASIC since it would not load in a COMAL program):

	1571	1581
120 Block BASIC file		
Load:	17 secs	8 secs
Save:	74 secs	31 secs
74 Block COMAL file		
Load:	48 secs	42 secs
Save:	60 secs	27 secs

[*Editors note: we got a note from Alex Jackson saying "The 1581 is the best thing Commodore has ever done (really fast)". We also got the latest Big Blue Reader 128/64. It can transfer files between 1541, 1571,* **and 1581** *drives in Commodore, CP/M and IBM disk formats (yes, even 3.5" 720K MS-DOS disks). $44.95 from SOGWAP, 115 Bellmont Rd, Decatur, IN 46733. The 1581 is supported only with a C128.*]

more»

```
// save "partition aid" // by Gary Parkin 1-88
// this program helps in making partitions
line$:=""17"Would you like to "
main
//
FUNC yes
  IF yn$="Y" OR yn$="y" THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
ENDFUNC yes
//
PROC center(text$)
  screen:=40
  length:=LEN(text$)
  space'left:=((screen-length) DIV 2)
  PRINT SPC$(space'left),text$
ENDPROC center
//
PROC main
  REPEAT
    PAGE
    PRINT ""14""
    center("* * PARTITION PROGRAM * *")
    PRINT
    PRINT "This Program helps in making"
    PRINT
    PRINT "Partitions with the 1581 Drive."
    PRINT
    PRINT
    PRINT
    PRINT "          "18"F"146"ormat Disk"
    PRINT "          "18"C"146"reate Partition"
    PRINT "          "18"S"146"elect Partition"
    PRINT "          "18"D"146"irectory"
    PRINT "          "18"Q"146"uit"
    PRINT
    PRINT
    INPUT "Enter Command (F/C/S/D/Q): ":
    comm$//wrap line
    IF comm$ IN "FfCcSsDdQq" THEN
      CASE comm$ OF
      WHEN "F","f"
        PRINT line$
        INPUT "Format a disk? (y/n) :": yn$
        IF yes THEN format
      WHEN "C","c"
        PRINT line$
        INPUT "Create a partition? (y/n):": yn$
        IF yes THEN create'partition
      WHEN "S","s"
        select'partition
      WHEN "D","d"
        PAGE
        DIR "2:"
        INPUT ""17"Press <return> to continue":
        yn$//wrap line
      WHEN "Q","q"
        INPUT ""17"Would you like to Quit?
        (y/n) :": yn$//wrap line
        IF yes THEN END "Bye ..."
      ENDCASE
    ENDIF
  UNTIL TRUE=FALSE
ENDPROC main
//
PROC format
  PAGE
  center("* * FORMAT DISK * *")
  PRINT
  PRINT "Place Blank disk in Drive #2"
  PRINT
  INPUT "and PRESS RETURN: ": yn$
  PRINT
  INPUT "Enter Disk Name :": diskname$
  IF diskname$<>"" THEN
    PRINT
    INPUT "Enter ID :": id$
    IF id$="" THEN id$:="xx"
    PRINT
    PRINT "ALL FILES WILL BE DELETED!"
    PRINT
    INPUT "OK? ": yn$
    IF yes THEN
      PASS ("n0:"+diskname$+","+id$),9
      PRINT
      PRINT STATUS$
      FOR t:=1 TO 5000 DO NULL
    ENDIF
  ENDIF
ENDPROC format
```

more»

```
//
PROC create'partition
  PAGE
  center("* * CREATE PARTITION * *")
  REPEAT
    PRINT
    INPUT "Enter starting track (1-39,
    41-80) :": track//wrap line
  UNTIL (track>0) AND (track<>40) AND
  (track<81)//wrap line
  PRINT
  REPEAT
    PRINT
    PRINT "To be a Sub-Directory, blocks must"
    PRINT
    PRINT "be a multiple of 40 and at least
    120"//wrap line
    PRINT
    PRINT
    INPUT "Enter number of Blocks :": blocks
  UNTIL (blocks MOD 40=0) AND (blocks>0)
  PRINT
  INPUT "Enter Partition Name :": part'name$
  IF part'name$<>"" THEN
    a1:=INT(blocks/256)
    b1:=blocks-a1*256
    PRINT
    INPUT "OK to create partition?(y/n) :": yn$
    IF yes THEN
      PRINT
      PRINT "CREATING PARTITION"
      PASS ("/0:"+part'name$+","+CHR$(track)+
      ""0""+CHR$(b1)+CHR$(a1)+",c"),9//wrap
      PRINT
      PRINT STATUS$
      PRINT
      PRINT line$+"make a"
      INPUT "Sub Directory? (y/n) :": yn$
      IF yes THEN format'partition
    ENDIF
  ENDIF
ENDPROC create'partition
//
PROC select'partition
  PAGE
  center("* * SELECT PARTITION * *")
  PRINT
  INPUT "Enter Partition Name :
  ROOT"157""157""157""157"": part'name$//wrap
  IF part'name$<>"" THEN
    IF part'name$="ROOT" THEN
      PASS "/",9
    ELSE
      PASS ("/:"+part'name$),9
    ENDIF
    PRINT
    PRINT STATUS$
    FOR t:=1 TO 5000 DO NULL
  ENDIF
ENDPROC select'partition
//
PROC format'partition
  PAGE
  center("* * FORMAT PARTITION * *")
  PRINT
  INPUT "Enter Partition Name :": dirname$
  IF dirname$<>"" THEN
    PRINT
    INPUT "Enter ID :": id$
    IF id$="" THEN id$:="xx"
    PRINT
    INPUT "OK to Create Directory? ": yn$
    IF yes THEN
      PRINT
      PRINT "CREATING DIRECTORY"
      PASS ("/:"+part'name$),9
      PASS ("n0:"+dirname$+","+id$),9
      PRINT
      PRINT STATUS$
      FOR t:=1 TO 5000 DO NULL
    ENDIF
  ENDIF
ENDPROC format'partition
```

Cougos Clipper had time test results for a 120 block BASIC file load reported by Greg Rogers:

1541/c64	*120 secs*
1581/c64	*60 secs*
1571/c128	*12 secs*
1581/c128	*8 secs* ■

Task Suggestion

by Borge Christensen

[*Borge founded COMAL back in 1974. He is still trying to make COMAL the best it can possibly be. Here is his latest suggestion.*]

Concurrent COMAL. A suggestion sketched.

It is suggested that COMAL is extended with the following structure:

TASK «*name*» [(«*list of qualifiers*»)]
  «*body of task*»
ENDTASK [«*name*»]

The body may consist of the usual COMAL statements.

The structure defines a task which may be executed parallel to other tasks.

The qualifiers are like parameters, only a meaning is assigned a priority. The set of possible qualifiers has not yet been finally defined. From an industry viewpoint, it is suggested that qualifiers might give the priority of a task, whether it is running or stopped at program start, etc. The whole subject is left for the development group to discuss.

Communication between tasks is undertaken by the statements:

SEND [«*p1*» **TO**] «*mailbox*»

and

AWAIT [«*p2*» **FROM**] [«*mailbox*»] [**TIME** «*time*»]

where:

«*p1*» ::= «*actual parameter*»
«*p2*» ::= «*formal parameter*»

To start and stop tasks, the following statements are suggested:

STOPTASK

and

STARTTASK «*name of task*»

STOPTASK is only active inside a task.

More about the structure and the statements is shown in the example below. A one armed robot is working with a conveyor belt. The driver for the controller is a machine coded package called ROBOX. In this case the controller is a small LEGO-box with six outputs, 0-5, and two inputs, 6-7.

```
USE robox
//
TASK arm CLOSED // no qualifiers
  // no own variables
  LOOP
    AWAIT object'ready
    get'object
    turntable'forward(170)
    AWAIT belt'ready
    let'go'object
    turntable'backward(10)
    SEND go'belt
    turntable'backward(160)
  ENDLOOP
  //
  // nested PROCs inside TASK arm
  //
  PROC turntable'forward(x)
    set'robox(2)
    SEND x TO counter'7
    AWAIT countdown FROM counter'7
    reset'robox(2)
  ENDPROC turntable'forward
  //
  PROC turntable'backward(x)
    set'robox(3)
    SEND x TO counter'7
    AWAIT countdown FROM counter'7
    reset'robox(3)
  ENDPROC turntable'backward
```

more»

```
  //
  PROC get'object
    set'robox(0)
    AWAIT TIME 0.25
    reset'robox(0)
  ENDPROC get'object
  //
  PROC let'go'object
    set'robox(1)
    AWAIT TIME 0.25
    reset'robox(1)
  ENDPROC let'go'object
  //
ENDTASK arm
//
TASK belt CLOSED // no qualifiers
  //no own variables
  LOOP
    TRAP
      AWAIT go'belt
      start'belt
      AWAIT watchdog'6 TIME 10
      stop'belt
      AWAIT TIME 5
      start'belt
      AWAIT TIME 1
      stop'belt
      SEND belt'ready
    HANDLER
      PRINT "Belt Timeout. Inspect system."
      PRINT "Press <<F7>> to reactivate."
      stopsystem
    ENDTRAP
  ENDLOOP
  //
  // belt's own procedures:
  //
  PROC start'belt
    set'robox(4)
  ENDPROC start'belt
  //
  PROC stop'belt
    reset'robox(4)
  ENDPROC stop'belt
  //
ENDTASK belt
```

Inside the main loop of the task **arm** a message from the mailbox **object'ready** is awaited. As soon as such an unspecified message is received, the object is grabbed and the robot arm is turned forward until the object is over the conveyor belt. Now a message from the mailbox **belt'ready** is awaited. Then the object is dropped unto the belt and the arm is turned aside. A message is then sent to the mailbox **go'belt**, and the arm is sent back to where the objects may be fetched.

The procedures of the task should be self-explanatory. The task belt has a built in TRAP structure. This TRAP is activated by the statement

AWAIT watchdog'6 TIME 10

if more than 10 seconds elapse before a signal from the sensor (watchdog) connected to input 6 is sensed. Industry people consider this use of the TRAP structure as very important.

The suggestions sketched above are made after many discussions with Ib Havn and Erik Ronhoj from Danish System Industry, Horsens and Jorgen Staunstrup and his students Jens B Hansen, Jacob Kornerup, and Soren Laursen, University of Aarhus.

I admit that the description is in no way final. I hope it can be further discussed at our next COMAL Standards meeting.

Further Reference:

COMAL Kernal, #17 pg 40
How to Use the Kernal, Richard Bain, #17 pg 55
■

IBM Windows

by Tom Kuiper

[*This is most of the letter that accompanied the IBM COMAL Windows program we received. It looks like we may now have enough people with IBM PC COMAL that we can put out a few program disks. The disks will be called* **Special Series** *... and Special Series #1 will include this program along with a couple converter programs that convert C64 Paperclip or C64 SEQ files into Word Perfect files and more.*]

The new COMAL looks very good. It seems that all the kernel and graphics package features have been implemented, and there are some new keywords. I like **AND THEN** and **OR ELSE**:

IF a>=0 AND THEN SQR(a)=2 THEN PRINT "A must be 4"//*wrap line*

does not cause an error if **a<0**. Likewise, the following does not cause an error:

IF a<0 OR ELSE SQR(a)=2 THEN PRINT "A is 4 or negative"//*wrap line*

RETRY can be used in the HANDLER section of an error trapping structure to re-execute the code that originally caused the error.

RENUM has been expanded to allow a selected range (or PROC or FUNC) to be renumbered.

MERGE can now insert the merged code aat a specified line number. These are very handy for re-organizing programs.

There is a sound package, and the data communications package looks very versatile. Instructions on creating your own packages seem clear, although I haven't tried it yet.

One problem seems to be the ARCL procedure in the GRAPHICS package. I get an OVERFLOW error. Pass it on! ARCR seems to work all right.

I'm enclosing a set of windowing procedures (WINDOWS.LST) along with a test program (TESTWIND.LST). These are on an IBM formatted disk, of course. You'll need to ENTER the first file and MERGE the second file.

WINDOWS.LST has the following procedures:

MAKE_WINDOWS(number#) defines the stack needed for the number of windows you plan to use. You call it once at the beginning of the program. The stack holds data about the previously opened windows.

OPEN_WINDOW(row, column, height, width, border_style#, label$, bkgnd_clr, border_color, text_color) opens the window:

row	row number for upper left corner
column	column no. for upper left corner
height	height of window, <= 24-row
width	width of window, <= 81-column
border_style#	type of border =1, single line =2, double line =3, solid border
label$	window label, length <= width-2
bkgnd_clr	0 .. 7
border_color	0 .. 15
text_color	0 .. 15

WINDOW_BORDER(row, height, column, width, border_style#, border_color, bkgnd_clr, border_color, text_color) actually does the work of making the labelled border and can be used separately from the windowing routines to make nice boxes on your screen.

CLOSE_WINDOW closes the window last opened.

On a different topic, I don't think that I ever told you that I had run your SIEVE benchmark on two other machines. Here are the results:

	Not Printing	Printing
Compaq Portable 286	2.2	3.8
HP Vectra	2.25	5.6

more»

IBM WINDOWS Program

```
USE system
make_windows(3)
define_colours
open_window(1,1,25,80,1,"This is window 1",
black,white,l'grey) //wrap line
DIR
PRINT "Stack pointer=";window_stk_ptr#
wait_for_key
open_window(10,2,5,78,2,"This is window 2",
l'grey,white,white) //wrap line
DIR
PRINT "Stack pointer=";window_stk_ptr#
wait_for_key
open_window(16,9,8,68,3,"This is window 3",
black,l'grey,white) //wrap line
DIR
PRINT "Stack pointer=";window_stk_ptr#
wait_for_key
close_window
PRINT "Stack pointer=";window_stk_ptr#
wait_for_key
close_window
PRINT "Stack pointer=";window_stk_ptr#
wait_for_key
close_window
PRINT "Stack pointer=";window_stk_ptr#
END
//---------------------------------------
PROC wait_for_key
  PRINT "Press any key to continue"
  INPUT "": answer$,
ENDPROC wait_for_key
//
PROC define_colours
  black:=0; d'grey:=8; d'blue:=1; blue:=9
  red:=4; pink:=12; purple:=5; lilac:=13
  green:=2; l'green:=10; cyan:=3; br'cyan:=11
  gold:=6; yellow:=14; l'grey:=7; white:=15
ENDPROC define_colours
//---------------------------------------
PROC make_windows(number#)
  DIM screen$(number#) OF 4006
  // screen before new opening window
  DIM window_stk#(0:number#,5)
  // previous windows defined
  window_stk#(0,1):=1 // top row
  window_stk#(0,2):=25 // bottom row
  window_stk#(0,3):=1 // left column
  window_stk#(0,4):=80 // right column
  window_stk#(0,5):=curattr# // colors
  window_stk_ptr#:=0 // stack pointer
  window_stk_max#:=number# // size of stack
ENDPROC make_windows
//
PROC open_window(row,column,height,width,
border_style#,label$,bkgnd_clr,border_color,
text_color) CLOSED //double wrap line
  IMPORT curattr#,getscreen,textwindow
  IMPORT textcolor,setattr // SYSTEM
  IMPORT putchar,set_text_attribute
  IMPORT window_border,window_stk_ptr#
  IMPORT window_stk_max#,window_stk#(,)
  IMPORT screen$()
  IF window_stk_ptr#<window_stk_max# THEN
    window_stk_ptr#:+1
    textwindow(1,25,1,80) //save current screen
    getscreen(screen$(window_stk_ptr#))
    window_border(row,height,column,width,
    border_style#,border_color,bkgnd_clr,
    label$) //double wrap line
    textwindow(row,row+height-1,column,
    column+width-1) //wrap line
    //define area inside border as real window
    textwindow(row+1,row+height-2,column+1,
    column+width-2) //wrap line
    CURSOR 1,1
    textcolor(text_color,FALSE,bkgnd_clr)
    PAGE
    window_stk#(window_stk_ptr#,1):=row+1
    window_stk#(window_stk_ptr#,2):=row+
    height-2 //wrap line
    window_stk#(window_stk_ptr#,3):=column+1
    window_stk#(window_stk_ptr#,4):=column+
    width-2 //wrap line
    window_stk#(window_stk_ptr#,5):=
    text_color+16*bkgnd_clr //wrap line
  ELSE
    PRINT "Window stack is full. Press a key."
    WHILE KEY$="" DO NULL
    textwindow(1,25,1,80)
```

more»

```
    set_text_attribute(window_stk#(0,5))
    PAGE
    STOP
  ENDIF
ENDPROC open_window
//
PROC window_border(row,height,column,
width,border_style#,border_color,
bkgnd_clr,label$) CLOSED //double wrap line
  IMPORT textcolor,putchar
  DIM top'left#(3), top'right#(3)
  DIM bottom'left#(3), bottom'right#(3)
  DIM top#(3), bottom#(3), vertical#(3)
  // single line border
  top'left#(1):=218; top'right#(1):=191
  bottom'left#(1):=192
  bottom'right#(1):=217; top#(1):=196
  bottom#(1):=196; vertical#(1):=179
  // double line border
  top'left#(2):=201; top'right#(2):=187
  bottom'left#(2):=200
  bottom'right#(2):=188; top#(2):=205
  bottom#(2):=205; vertical#(2):=186
  top'left#(3):=219; top'right#(3):=219
  bottom'left#(3):=219
  bottom'right#(3):=219; top#(3):=223
  bottom#(3):=220; vertical#(3):=219
  // solid border
  textcolor(border_color,FALSE,bkgnd_clr)
  attr#:=bkgnd_clr*16+border_color
  putchar(top'left#(border_style#),row,
  column,attr#) //wrap line
  CURSOR row,column+1
  left_part_end#:=(width-LEN(label$))/2
  FOR i#:=2 TO left_part_end# DO PRINT CHR$
  (top#(border_style#)), //wrap line
  PRINT label$,
  right_part_start#:=left_part_end#+
  LEN(label$)+1 //wrap line
  FOR i#:=right_part_start# TO width-1 DO
  PRINT CHR$(top#(border_style#)), //wrap line
  putchar(top'right#(border_style#),
  CURROW,CURCOL,attr#) //wrap line
  FOR i#:=row+1 TO row+height-2 DO
    putchar(vertical#(border_style#),i#,
    column,attr#) //wrap line
    putchar(vertical#(border_style#),i#,
    column+width-1,attr#) //wrap line
  ENDFOR i#
  putchar(bottom'left#(border_style#),row+
  height-1,column,attr#) //wrap line
  CURSOR row+height-1,column+1
  FOR i#:=2 TO width-1 DO PRINT CHR$(
  bottom#(border_style#)), //wrap line
  putchar(bottom'right#(border_style#),
  CURROW,CURCOL,attr#) //wrap line
ENDPROC window_border
//
PROC close_window CLOSED
  IMPORT screen$(),window_stk#(,)
  IMPORT window_stk_ptr#,set_text_attribute
  IMPORT setscreen,textwindow
  set_text_attribute(window_stk#(
  window_stk_ptr#-1,5)) //wrap line
  PAGE
  textwindow(1,25,1,80)
  setscreen(screen$(window_stk_ptr#))
  window_stk_ptr#:-1
  textwindow(window_stk#(window_stk_ptr#,
  1),window_stk#(window_stk_ptr#,2),
  window_stk#(window_stk_ptr#,3),
  window_stk#(window_stk_ptr#,4))//4wrap lines
ENDPROC close_window
//
PROC set_text_attribute(attr#) CLOSED
  IMPORT textcolor,setattr
  blink#:=attr# DIV 128
  backg#:=(attr# MOD 128) DIV 16
  text#:=attr# MOD 16
  // PRINT blink#;backg#;text# // test
  textcolor(text#,blink#,backg#)
ENDPROC set_text_attribute
//
PROC putchar(ascii#,row,col,attr#) CLOSED
  IMPORT setattr,setchar
  CURSOR row,col
  setattr(attr#)
  setchar(CHR$(ascii#))
ENDPROC putchar ■
```

Apple COMAL Notes

by David Stidolph

For the last year and a half I have been developing COMAL for the Apple II family of microcomputers. I say family because there have been at least 5 major versions released. In order of their release they are:

Apple][
Apple][+
Apple //e
Apple //c
Apple IIgs

Since the introduction of the Apple][over 10 years ago, Apple has made every effort to increase the power and capability of their system without sacrificing compatibility. In large part they have succeeded. The major hardware features of the Apple are:

- Expandable through the use of "slots"
- Built in BASIC language
- 40 Column video screen
- Lo-Res color graphics using the text screen
- Hi-Res color graphics using 8k of memory

Apple expanded the system with the //e model. It has added memory capability to 128k and an 80 column screen.

Disk access has similarly evolved on the Apple. Once a simple disk system for loading files, restricted to 5.25" drives and slow access, the new disk operating system (called ProDos) supports subdirectories and a variety of disk sizes.

System Definition for Apple COMAL

Before I started to write Apple COMAL, I first decided that the minimum system it should run on would be the Apple][+ or later with at least 64k of memory and AppleSoft BASIC in ROM. This means COMAL should work on almost every Apple in the schools.

In addition to the minimum system I felt there were certain features I should support if they were available:

Lower Case characters
80 Column screen
RAM disk
Joystick
Paddles

File Access Under Apple COMAL

I have made every effort to make Apple COMAL compatible with the C64 version of COMAL. FILE access is one such area. The device names used by the COMAL 2.0 cartridge are supported, along with some new names to make life easier for beginners.

Name	Device
LP:	Slot 1
DS:	Screen
SP:	Slot 2
KB:	Keyboard
0:	Disk Drive
...	
7:	Disk Drive
PRINTER:	Slot 1
SCREEN:	Video Screen
MODEM:	Slot 2
KEYBOARD:	Keyboard
A:	Disk Drive
...	
G:	Disk Drive
NULL:	Null (no) Device

The first group of file names are those supported by the COMAL 2.0 cartridge. I have implemented them and set their defaults to the standard set used with the Apple //c. (Most Apple software, for example, assumes your printer is connected to slot number 1 - I make the same assumption).

more»

The second group of device names are those I have added for ease of use. Disk drive names **A:** through **G:** have been added for those coming from an IBM or CP/M background. The term **PRINTER:** is easier to remember than **LP:** (at least for me). This does not mean you **have** to use the new names - they are there only for your convenience. You may use any you choose.

The last name on the second list is new to Commodore and Apple users - **NULL:**. This device is literally NO device. OUTPUT to this device is sent nowhere and INPUT from it is nothing (**CHR$(0)**). **EOF** on the **NULL:** device is always **TRUE.**

Most file commands will now work the same on both machines. For example, to LIST the program in memory to the printer you would use the command:

LIST "LP:"

This is the same command you would use with Commodore COMAL - so the instructions to list the program in memory to the printer would be the same for both.

Setting Linefeeds and High Bits

I have also added file attributes to filenames. They are signaled by commas (",") after the filename. They are:

,L+ Send Linefeed after carriage return
,L- Do not send linefeed after carriage return
,H+ Set high bit
,H- Clear high bit

To list the program in memory to the printer with linefeeds and high bit clear (settings would differ for different printers) you could type:

list "printer:,l+,h-"

Listing a Disk Directory

The **CAT** and **DIR** commands have been enhanced to provide the information you need - in a format designed for 40 columns. The following is an example directory:

DISK NAME:/comal/programs/

TYPE	FILENAME	MOD DATE	BLKS
SYS	"PRODOS"	!01/12/87	32
SYS	"COMAL.SYSTEM"	!03/15/88	76
CML	"STARTUP"	!03/22/88	12
TXT	"FRUSTRATION"	!11/12/87	14
TXT	"README.TEXT"	!03/18/88	20

The directory is displayed in such a way that you may cursor up to a file name and type the commands RUN, LOAD, CHAIN, and DELETE followed by «*return*». The exclamation point is there so that the rest of the line (after the file name) is ignored (! and // signify comment).

Error Messages

One of COMAL's best features has been its error messages. Most languages do not report errors until compile/run time. COMAL automatically compiles a line as it is entered and will report syntax problems then. Apple COMAL now has very complete error messages. For example, consider the following line:

0010 PRINT FILE 5; inventory#

This line is incorrect because the semi-colon following the **5** should actually be a colon. Apple COMAL would report this error by placing the cursor on the semi-colon and printing on the line beneath:

":" EXPECTED, NOT ";"

This type of error message is much more complete than most other languages. ■

File Name Conventions

We try to include more than just COMAL programs on our disks. To avoid confusion, we've adopted naming conventions. Use these conventions to avoid unnecessary confusion over files, especially if you are submitting them to us.

All computer systems, up to 8 character NAME:

Name	Meaning (all systems)
NAME	COMAL program (IBM & CP/M automatically tack on their own extension)
NAME.LST	Listed program / proc / func
NAME.DSP	Displayed program
NAME.EXT	External procedure
NAME.DAT	Data file
NAME.TXT	Text file
NAME.DOC	Documentation file
NAME.BAS	Basic program
NAME.RAN	Random file

Commodore only ... up to 16 characters:

Suffixed	Prefixed	Meaning (C64/C128)
NAME.PROC	PROC.NAME	PROC listed to disk
NAME.FUNC	FUNC.NAME	FUNC listed to disk
NAME.DAT	DAT.NAME	Data file
NAME.TXT	TXT.NAME	Text file
NAME.DOC	DOC.NAME	Documentation file
NAME.EXT	EXT.NAME	External proc/func
	SHAP.NAME	Sprite shape file (for **loadshape**)
	IMAG.NAME	Sprite shape file (for **read file**)
	FONT.NAME	Font file (for **loadfont**)
	SET.NAME	Basic type font file
NAME.BAT	BAT.NAME	Batch file
	SNG.NAME	Song file
	HRG.NAME	Color COMAL picture
NAME.HRG		Black/White bitmap
	CRG.NAME	Compact Color picture
NAME.CRG		Compact B/W picture
	ICON.NAME	Print Shop Icon file
	SCRN.NAME	Text Screen file
	POP.NAME	Mergeable Popover
NAME.POP		Program with Popover
	CHIP.NAME	Super Chip Program ■

How To Type In Programs

Line numbers are required for **your** benefit in editing a program (but are irrelevant to a **running** program). Thus line numbers usually are omitted when listing a COMAL program. It is up to YOU to provide the line numbers. Of course, **COMAL can do it for you.** Follow these steps to enter a COMAL program:

1) Enter command: **NEW**
2) Enter command: **AUTO**
3) Type in the program. When done:

   Power Driver (0.14): Hit «*return*» key twice
   C64 / C128 COMAL 2.0 cart: Hit «*stop*» key
   CP/M COMAL 2.10: Hit «*esc*» key
   IBM PC COMAL: «*control*»+«*break*»
   Mytech: «*control*»+«*C*»

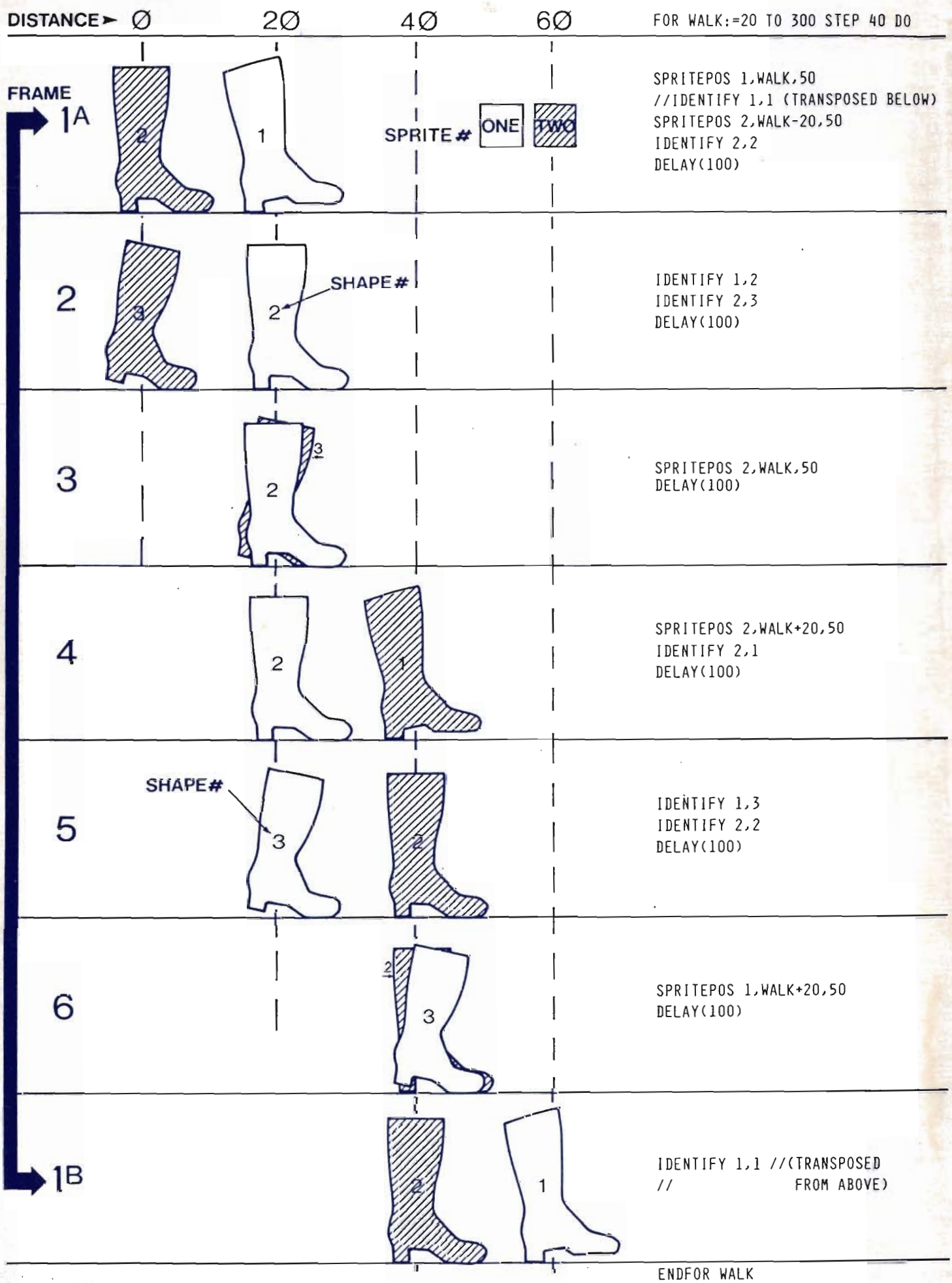
You may use both UPPER and lower case letters while entering a program (C64 0.14 users should upgrade to Power Driver). COMAL automatically makes keywords UPPER case and variable names lower case. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures, and COMAL will insert them for you. You **DO** have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line, we will continue it onto the next line and add *//wrap* at the end (and often print it in *italic type*). **You** must type it as one continuous line.

Variable names, procedure names, and function names can be a combination of:

abcdefghijklmnopqrstuvwxyz 0123456789'] [\ _

The «*left arrow*» key in the upper left corner of the C64/C128 keyboard is valid. COMAL 2.0 converts it into an underline. The C64/C128 computers use a «*British pound*» symbol in place of the \ backslash. ■


DISTANCE ► Ø
20
40
60
FOR WALK:=20 TO 300 STEP 40 DO
FRAME
1A
SPRITE#
ONE
TWO
SPRITEPOS 1,WALK,50
//IDENTIFY 1,1 (TRANSPOSED BELOW)
SPRITEPOS 2,WALK-20,50
IDENTIFY 2,2
DELAY(100)
2
SHAPE#
IDENTIFY 1,2
IDENTIFY 2,3
DELAY(100)
3
SPRITEPOS 2,WALK,50
DELAY(100)
4
SPRITEPOS 2,WALK+20,50
IDENTIFY 2,1
DELAY(100)
5
SHAPE#
IDENTIFY 1,3
IDENTIFY 2,2
DELAY(100)
6
SPRITEPOS 1,WALK+20,50
DELAY(100)
1B
IDENTIFY 1,1 //(TRANSPOSED
// FROM ABOVE)
ENDFOR WALK